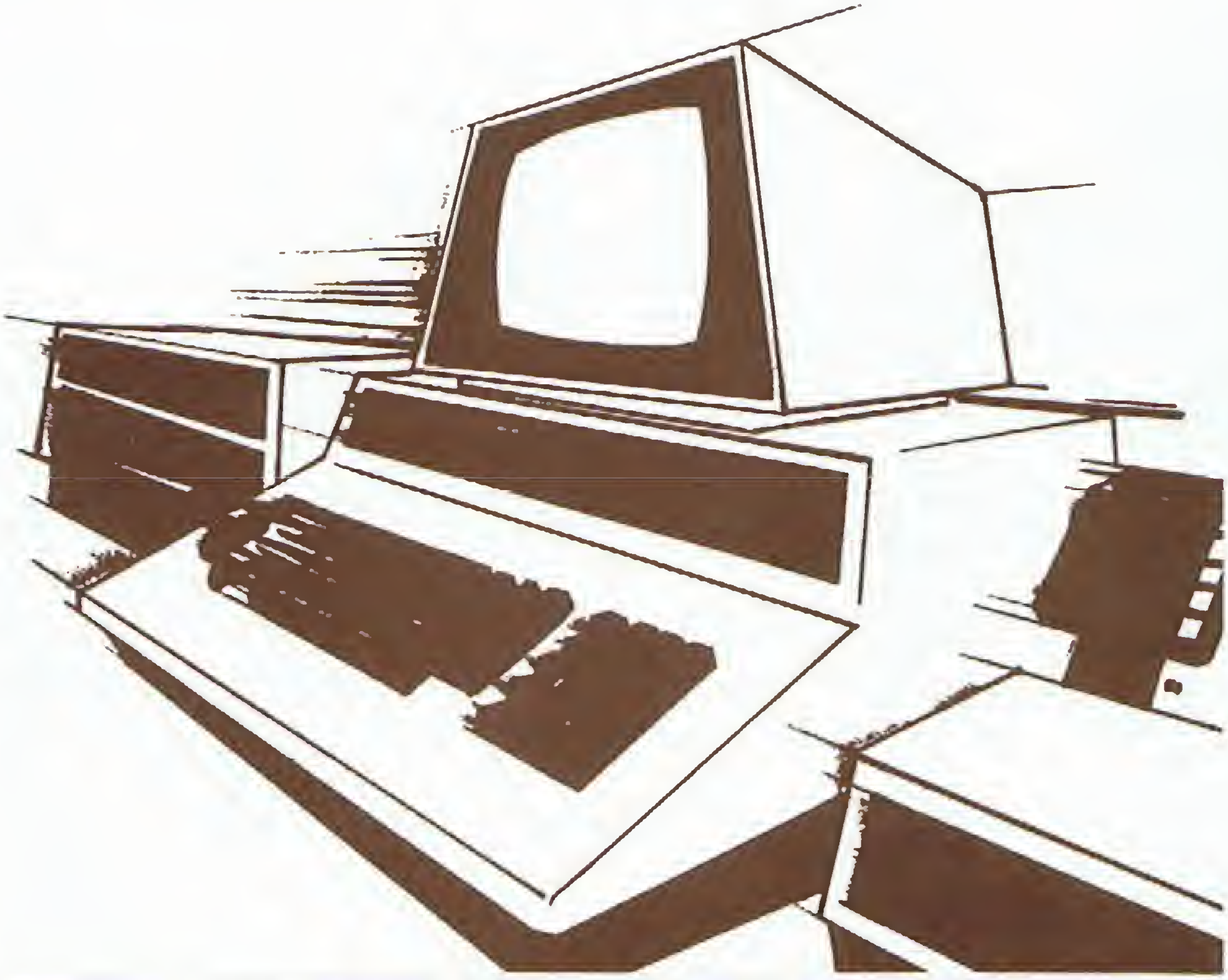
CPUCN

The Official Commodore Pet Users Club Newsletter



Volume 2

Issue 7

commodore

Contents

Commodore News

By Andrew Goltz

AFTER THE SHOW

Now that the dust has settled down after the tremendous success of the World's First PET Show, and CPUCN has returned to its usual format, it seems a good idea to review the present position. Several thousand people packed the main ballroom at the Cafe Royal and all exhibitors agreed that their participation had been extremely worthwhile. It seems likely that Commodore will be making the PET Show an annual event, if so then the 1981 event will have to be moved to a more spacious venue! It is unfair to make a selection from the many varied exhibits on the fifty or so stands but special mention should be made of the following: - the BMB Compuscience MUPET system (now being distributed in the UK by Kobra Systems through the Commodore Dealer Network.) - the new Super PET functioning as a remote terminal - and as a wordprocessor running WordPro 4. The position as regards the availability of the 80 column machines is that currently each of the Commodore Commercial Systems Dealers is receiving a single machine for training and familiarisation purposes and dealer's engineers are busy attending special 8000 series engineering courses being run by the Commodore Training Department.

It is expected that the first 8000 series machines will be delivered to users in about 60 days time, which is also the period within which a series of software application packages, currently under development, will be released.

The PET Show User Club seminars also provided a forum in which members could meet Commodore staff, and a number of outside speakers gave first rate presentations. A number of user suggestions are being implemented, and one that readers will immediately notice is the new CPUCN UPDATE section in support of Commodore's Business Software.

USER CLUB MANAGER APPOINTED

I am pleased to welcome David Middleton who has joined Commodore as User Club Manager. David is a Mechanical Engineer with extensive BASIC/Machine Code experience. Having run his own PET user group, he has a special interest in assisting local user groups solve the problems of getting off the ground. David has been heavily involved in the editing of this edition of CPUCN and will be taking over the editorship completely as from the first issue of Volume 3.

Whilst it is sad to hand over a job which has brought me a great deal of satisfaction it is rewarding to know that the editorial reins are being handed over to someone who has a keen interest in developing and extending the service offered to User Club members. I will not be leaving CPUCN completely as in the future I will be contributing application stories. I will also be involved with the development of sister publications like MISAC, a copy of which is enclosed. Although MISAC is intended primarily for use in schools and colleges a copy is being sent to all User Club members as part of our policy of keeping you up to date with all the latest Commodore literature, and I feel that much of the material (especially the Sheridan College article) will be of general interest to users outside the field of education. After you have finished with MISAC why not hand it over to a teacher friend.

One of the most popular speakers at the PET Show seminars was Jim Butterfield, who I would like to personally thank for flying over to the UK at extremely short notice. Jim brought over with him a number of excellent programs which he has put into the public domain. These include CROSS REF and IEEE WATCH which are being reproduced in CPUCN, an improved edition of SUPERMON and a superb version of ADVENTURE which runs on a 32K PET and disk unit! David Middleton will be bringing these programs with him when he visits local user groups and members will be free to make their own copies. Please write to David if your group is interested.

Update

ATTENTION COMPAY & WORDPRO III USERS

Recent batches of PETs have somewhat changed the balance of our video circuitry. The net result is that the fast screen poke has become extremely unreliable to use.

The fast screen poke is designed to speed up the print speed on the PET's screen. There are two versions of it, as follows:

	Poke Location	Fast	Slow
1.	59458	60	30
2.	59490	62	30

Version 1. is used in Comwordpro III, and version 2. is used in Compay. In the light of recent hardware developments, both versions must now be classed as OFFICIALLY NOT SUPPORTED.

Compay updates, effective June 1 1980, include removal of the fast screen poke. Comwordpro III is now being shipped without the poke as well. If you have an old version of WP3 and are experiencing difficulties, do the following:

1. Load Comwordpro III, DON'T TYPE RUN.

2. POKE 5348,234:POKE 5349,234:POKE 5350,234

3. Re-save Comwordpro III.

This will remove the fast screen poke from your version of Comwordpro III.

M J R Whitehead

Petpacks

As you will be aware from the advertisement on the back page of our previous newsletter, a company called Audiogenic (tel. Reading [STD CODE 0734] 595269) is handling mail order requests for all Commodore cassette software, and a range of the disk products as well. A growing number of 'add-ons' for the Pet are also available from them, but the main reason for mentioning them is that they are able to distribute copies of our software catalogue, and the new one has just been published. I'd like to draw your attention to this catalogue, as we have announced in it a large number of new cassette titles for the PET. These cover a wide range of applications from word proccessing to gardening, from utilities to languages, and from entertainment to education.

The last two in particular have seen a great leap in quantity and particularly quality over the last few months, as more and better programs have become available. Our Arcade series of games has produced three wonderfully entertaining and very addictive games, and all three show what can be achieved graphically by the use of machine code. On the education side, there are nine new titles in the current catalogue, and these too show what an increase there has been in the standard of programming since the PET first appeared.

I don't apologise for boasting about these programs - after all, we didn't write them! Our only achievement is in selecting and distributing the best out of a very large number of programs that are written for the PET and to make sure that you don't have any worries when buying a program from Commodore.

But if we don't write them, who does ? The answer may well suprise you - you do! If it wasn't for the contributions of people like yourselves, PET devotees, there wouldn't be such a good library of programs for people to enjoy. So, people like Bob Chappell, Deri James, Ted Landsler, Andrew Colin, A. Russell Wills, and the rest, take a pat on the back. And if I didn't mention your name, it's only because space precludes it!

In order to maintain our standards, and to continue supplying value-for-money software, we obviously need a continuous supply of good programs. This is where you come in. This month I'd like to start a competition, which is, of course, to write a program, and there will be six categories, broken down into two games and four educational programs. These are :-

1) Simulation of a board game.

2) Arcade quality game (like Invaders etc).

3) Physics tutorial program.

4) Chemistry tutorial program.

5) Mathematics tutorial program.

6) Biology tutorial program.

Exactly what format these programs can take I'll leave up to you. Suffice it to say that we are looking for a very high quality. The closing date will be two months from the date you receive this newsletter, and the best program in each category will be published in our library of programs, and the winning authors will get a ten per cent royalty on the programs that we sell. If your program is reasonably succesful (and a good program usually is!), this could amount to something in the nature of 20 to 30 pounds a week. Can't be bad.

So, budding geniuses, let's see what you can do. Send your entries (on tape please) to :-

Pete Gerrard,
Cassette Library Manager,
Commodore Business Machines,
818 Leigh Road,
Trading Estate,
Slough,
Berks.

Full details of all the winning programs will be published in 3 newsletters time. Good luck, and I look forward to seeing your programs.

Book review

PET/CBM PERSONAL COMPUTER GUIDE
by Carroll S. Donahue and Janice K. Enger

This latest publication from Adam Osborne/McGraw Hill maintains the excellent standards established by their earlier classics such as 'An Introduction to Microcomputers and '6502 Assembly Language Programming .

The book is a well written, comprehensive guide to the PET that will be found useful by the novice, beginning programmer and experienced PET User. The material in the book being well signposted' to indicate the level of difficulty.

As expected in Osborne publications diagrams and worked examples are extensively used throughout the book and they successfully compliment the main text. A new departure is the use of photographs and screen displays which help to make the book 'friendly' for the absolute beginner.

Sometimes the book appears to be guilty of providing too much information e.g. the section of PET trivia includes a discussion of what happens when three keys on the PETs keyboard are depressed

simultaneously! But in most instanaces the reader is warned that he is reaching a section which may be skipped at first and often such sections provide invaluable information on PET quirks which provide the experienced PET programmer with useful reference material.
The book includes an introductory chapter that explains the basic concepts common to microcomputers and looks at PET's special features. The next chapter called 'Operating The PET' deals thoroughly with the installation, correct use and maintenance of the PET hardware. The next two chapters deal comprehensively with PET BASIC commands and statements and the techniques of writing, editing and saving programs. Chaper 5, entitled 'Making the most of PET features', discusses string handling, programmed cursor movement, graphics, animation and file handling. PET special features, including the graphics character set, internal clock and Poking screen memory, are well covered. The final chapter will be especially useful for advanced programmers as it deals with the details of PET's own operating system, and introduces the concept of Machine Code programming.

A comprehensive series of appendices provide memory maps and tables for BASIC 1.0 and BASIC 2.0, and an excellent bibliography completes a well thought out and carefully researched book. I have no hesitation in recommending a copy for every PET user's book-shelf.

The book may be ordered via your local Commodore Dealer or direct by post from Audiogenic Software, P.O. Box 88, Reading, Berks. Tel. 0734 595269.

(Also received 'PET and the GP IEEE/488 BUS' published by Adam Osborne/McGraw Hill which will be reviewed in CPUCN 2.8.)

Computer philosophy

W.T. Garbutt
Mississauga, Ont

"It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light , it was the winter of despair, we had everything before us, we had nothing before us, we were all going directly to Heaven, we were all going directly the other way--in short, the period was so much like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only."

So began Charles Dickens in the famous introduction of the 18th Century setting for "A Tale of Two Cities" over a century ago! As with all great literature, the vision Dickens gives is perhaps centuries ahead of its time, in any event just as germane today as it was when Dickens first wrote it. Today Western civilization is on a precipice; the brink of a great leap forward for Humanity or a momentary or perhaps fatal fall.

While less than one-sixth of the world's population consumes over 80% of world resources, hundreds of millions fight daily to survive on the remainder. And those who survive remember!

For the fortunate few, the leap forward has begun. While scholars will argue for generations as to the precise time, (there probably is not one) the race to the Moon will likely symbolize the start for many. None of the generation of the sixties will forget President Kennedy's proclamation to reach the Moon by the end of that decade (nor will they likely forget his tragic and senseless death). The last brief decade has witnessed a massive expansion of knowledge, the extent of which is beyond human comprehension. If all knowledge known were increased by the order of a magnitude in the 70's, is there any doubt the increase will be two orders of magnitude this decade? Is there any certainty that this exponential knowledge explosion will quench humankinds' thirst by the 21st century?

Yes. There is serious doubt if the knowledge does not bring with it a maturity, a compassion for fellow man, a sharing and a sacrifice . The dismal scenario has already been summarized by Mr. Dickens in the 18th century: war, pestilence, famine and destruction. And even with our present limited knowledge is there any doubt that a similar order of a magnitude, an exponential order of horror could soon be our legacy.

There are disquieting signs that the future optimism, our traditional western belief in historical progress may be open to serious question. What Western civilization must guard against is introspection. Cultural isolation will almost certainly provide the impetus for an apocalypse. This introspection can take many forms. Most familiar to the readers of The Transactor is the discipline of electronics and more particularly computers.

Certainly this is a paradox. The computer, the single most important element in the recent knowledge explosion, the genie that could provide the key to the crucial inter-relationships in what at present is an incomprehensible mass of data, surely could not be viewed as retrenchment. On the contrary if a limited number of well educated and affluent people become preoccupied with the computer solely for their own personal intellectual gratification that is precisely what could happen.

However it need not happen. What is required is an atmosphere of freedom, freedom of thought, expression, knowledge freely accessible to all as well as freedom from hunger and poverty. A vast educational challenge exists certainly as great as that witnessed in the race to the Moon. And the micro-computer is ideally suited as a central instrument in meeting the challenge.

One of the major micro-computer

applications will be communications. The micro-computer must become as universally used as the light bulb. It must be available to the 'outbacks' of civilization as well as the present temples of Western civilizations (Universities). Computer literacy must become, with literacy, the prime objective of man after the basic survival improvements over hunger and poverty. No longer must language create communication barriers between nations, no longer may state bureaucracies hide knowledge clandestinely or inadvertently, no longer can corporations shape consumer habits through controlled ignorance.

These freedoms of course carry risks. Change creates instability. Governments and citizens fear unbridled change especially anarchy. But the risks of not accepting the challenge is greater. We could lose all the significant advances that civilization to date has witnessed. The present world population could turn away from a Western tradition that failed to solve humankinds most immediate problems.

The computer can become a central tool to manage today's knowledge, freeing man's creative and underutilized imagination, permitting him time to examine himself.

Today Western civilization senses the future. One has only to witness the long lineups and multi-billion dollar box office receipts for 'science fiction' movies. Today's audiences are seeing implemented yesterday's dreams. And super realistic depictions of tomorrow. The audiences understand, as never before, the revolutionary technological concepts. Indeed they will accept nothing less. In many instances these special effects were only made possible with the aid of the computer. We must not permit this phenomenon to become a form of escapism. It is important to feed the scientific needs of our citizens. But we must also consider the equally important spiritual needs. eg. beliefs and values.

The computer can provide assistance to man. For every self-gratifying use there is equally a use that will help meet the challenges. From simple energy conservation methods such as controlling house temperature, or arranging car pools to complex techniques to improve work effiency computer uses abound to cut waste.

A recent industry census indicated there are over 500,000 computers in personal use today. Within two years personal ownership is projected to quintuple. Think of the immense number of applications such a user population can generate.

As a member of the micro-computer community, ask yourself the question "What uses can I derive and implement that will aid, no matter how insignificant it may appear, in meeting the challenges of today?" Talk to fellow users , share your ideas and you will likely find yourself in the vanguard of tommorrow's leaders.

Teach in

1. SAVING PROGRAMMES UNDER DEVELOPMENT.

When developing and testing programs using disk files there is a significant risk of a hang-up due to simple syntax errors. And if you haven't saved the program, then bang goes a few hours work straight down the drain!

In all my programs I have a standard subroutine called "SAVE PROGRAM UNDER DEVELOPMENT" or "SPUD"! located in lines 60000 to 60999: I write my programs in modules and after completion of each module, before testing, I save the program by executing a direct RUN 60000.

The routine does the following:-

a. Picks up the program title from a standard position in the first line of the program by peeking the memory. (Because I always lay out my programs in a standard way, I know the first character of the title will always be in the same location.)

b. Adds the first four digits of the time as held in TI$ to act as a generation number.

c. SAVES the program using the combined title and time.

d. VERIFIES the program.

The listing is shown at the end of this article.

2. SKELETON PROGRAMS

On several occasions, it has been pointed out how useful it is to lay out programs using blocks of line numbers in a standard way. After a while, you find yourself adopting the same line number blocks to perform similar functions whenever they appear in one of your programs. For example, in most of my programs using a menu to display options for the operator to select, the menu display is coded in block 2000-2999 and the selection routine in 2100-2199.

If you find you have adopted some similar convention of your own, why not set up a skeleton program, in order to save a fair amount of typing time at the start of each coding session. In this skeleton program will be the REM statements for the title and preliminary blocks plus the actual coding for all the standard subroutines which are likely to appear in most of your programs.

My own skeleton program runs to 1000

characters which I would have had to type in each time. It contains a Title block, a Preliminaries block and subroutines for Keyboard input, Get a valid character, Cash right align, Delay for 2.5 seconds and Save program under development.

The REM statements at the front of each block, and the underlinings using minus and equals signs take up a lot of space, but if you have the space spare, I recommend you use them or something similar. It makes listings so much easier to read.

In use, I load "SKELETON", then type in the title of the program being coded into the space reserved for it. From then on, as I finish typing in a module (block) of coding, I save it using SPUD in 60000.

```
READY.

 10 REM TITLE      :-"SKELETON    "
 20 REM WRITTEN BY:- M.J. GROSS-NIKLAUS
 30 REM FOR        :-
 40 REM STARTED ON:-
 50 REM LAST AMEND:-
 90 GOTO1000
 99 ========
 100 VARIABLES MAP
 101 --------
 110 A  - Z  = GENERAL PURPOSE
 111 (I,J,K  = G.P. LOOP VARIABLES )
 199 ========
 200 STRING VARIABLES MAP
 201 --------
 210 A$ - Z$ = GENERAL PURPOSE
 299 ========
 1000 REM PRELIMINARIES
 1001 REM --------
 1099 REM ========
 50000 REM AWAIT VALID KEY
 50001 REM--------
 50010 GETA$:IFA$=""THEN50010
 50020 FORZ=1TOLEN(Z$):IFMID$(Z$,Z,1)=A$THENRETURN
 50030 NEXT:GOTO50010
 50099 REM========
 50100 REM INPUT TRAP
 50101 REM--------
 50110 POKE158,3:POKE623,34:POKE624,34:POKE625,20:PRINTZ$;:INPUTA$:A=VAL(A$)
 50120 RETURN
 50199 REM========
 50300 REM 2.5 SECOND DELAY
 50301 REM--------
 50310 FORZ=1TO2500:NEXT:RETURN
 50399 REM========
 50400 REM CASH RIGHT ALIGN
 50401 REM--------
 50410 Z=INT(Z*100+.5)/100
 50420 Z$=LEFT$(RIGHT$("          "+STR$(Z+.005*SGN(Z)),10),9):RETURN
 50499 REM========
 60000 REM SPUD
 60010 T$="0:":FORI=1044TO1055:IFPEEK(I)=0THEN60030
 60020 T$=T$+CHR$(PEEK(I)):NEXTI
 60030 T$=T$+LEFT$(TI$,4)
 60040 PRINT"▮SAVING ▮";T$:SAVET$,8
 60050 VERIFYT$,8
READY.
```

3. A USEFUL TIP FOR PROGRAM LISTINGS.

Most printers have a double-line spacing facility yet it seems to be very rarely used for program listings. The improvement in legibility using double line spacing is very worthwhile. Perhaps more important, you can decide on amendments and write them in under or over the lines involved, without being at the PET.

When I started using the PET in early 1978, I did all my coding and development work at the keyboard, in direct contradiction to my main-frame training. Now the old disciplines have re-established themselves, I use double spaced listings all the time. I separate the sheets and put them into a clip-board, and work on them on train journeys for example. An added bonus is to sprawl outside on the grass on one of those rare sunny midweek afternoons, clip board and pen close at hand, commanding ones colleagues not to disturb you while you debug your program!

On the 3022 Commodore printer, you can open up the lines by a specified amount using a PRINT# to secondary address 6. The version I use is:-

```
OPEN1,4,6:PRINT#1,CHR$(45):CLOSE1
```

4. ENGINEERING TRAINING

The Training Department hasn't been the same since the first of May. On that day, Spiro Omfalos joined us from the Service Department. Using techniques which he has obviously adopted from the "Road Runner" films, typing with one hand, operating an oscilloscope with the other, while talking on the phone and listening to a cassette recording, he has put together one of the country's best micro-computer servicing courses in a period of four weeks. How do I know it's so good? Because that is what 90% of those present on the first course said, and backed it with maximum scores on our course appraisal sheets.

Service courses won't affect most of you personally, but it's nice to know that Commodore Dealer engineers have that kind of backup available.

Coming shortly will be conversion courses for the 8000 series equipment. Dealers, please contact Biddy Clark on 01 388 5702 for details.

5. ASSEMBLER COURSE.

Since we cancelled our assembly language (machine code) course due to lack of support, enquiries regarding such courses have built up. I intend to run a few this year possibly leading to a regular series next year.

The first one will be in late September/early October with Cranfield Institute of Technology as the probable venue for a three day residential course.

My intention is to make this a beginner's course, starting with micro-architecture, binary and hex, and leading to the point where you can confidently use the mini-assembler in EXTRAMON to write useful machine code routines. After attending the course, you should be able to teach yourself the full assembler from the manual.

Projected price is £275.00p inclusive of meals and accommodation but excluding VAT. Ring Biddy Clark on 01 388 5702 for details.

COMMODORE TRAINING DEPARTMENT

Assembly Language Course

Location: Cranfield Institute of Technology, Bedforshire.

Date: September 23 - 25 1980

Fee: £275.00 + VAT (inclusive of accommodation and meals.

Aim: To take you to the point where you can write useful machine code routines using the EXTRAMON mini-assembler and disassembler.

Start Point: You should have some familiarity with the concepts of programing (in BASIC or some other high level language).

Syllabus:
- Functioning of a micro-processor
- Main registers in the 6502 processor
- Holding information in Binary and Hex
- Addressing memory
- The fetch execute cycle
- The 6502 instruction set
- Using the EXTRAMON assembler and disassembler
- Binary addition and subtraction
- Boolean operations
- Addressing modes
- Linking to PET ROM routines
- Linking machine code routines into BASIC programs
- Coding for the User Port

Very friendly. Steady, thorough tutoring.

I find it extremely helpful to have tutors who are both practical and practising the subject matter of the course.

Thanks for the course. Your 'User Friendliness' rating is very high.

Very good, enjoyable course.

Full marks to your awareness of the particpant's requirements.

Course very well organised and well run with a good sense of humour! Many thanks, and you will probably see me again on an advanced course in the future — it was very good value for money.

The course met my requirements. I'd recommend it to others if the need arose.

Hard work but very valuable.

I em now much more confident of using the disk unit to its full potentiel.

COURSE BOOKING/ENQUIRY COUPON

I would like to book on/receive further details of the ASSEMBLY LANGUAGE COURSE, September 23 - 25 at Cranfield.

Name ....

Address ....

....

....

.... Tel:..

I enclose my cheque payable to Commodore Business Machines (UK) Ltd. in the sum of £...........

Please return this slip to: **Commodore Training Department**
360 Euston Road, London NW1 3BL

STRING HANDLING IN BASIC LANGUAGE

Most people are aware of a computer's calculating abilities. However, there is another side to computing that is not as familiar to the average person. We have all been exposed to computerised mailing lists, credit information, airline ticket reservations or some other example of the computers data handling capability. The purpose of this article is to relate how some of these things can be accomplished even using a microcomputer such as the Commodore PET.

Most microcomputers today come with BASIC as the major language. BASIC was designed to be a very simple language, easily learned, which can handle most of the operations that are associated with large computers. The penalizing aspect of BASIC is the speed of execution which is often not a major consideration in the use of fixed handling data. We will try to explain each and include examples.

Before we begin the discussion of the commands of BASIC, it is necessary to define some terms. We must understand how the computer stores its data as code numbers. The microcomputer is actually only capable of storing numbers in the range from 0 to 255. As a result of this some standard codes have been created. The code convention used by micros is called ASCII (American Standard for Computer Interchange of Information). For example, in ASCII code the letter "A" is represented by the number 65, "B" by 66, and so forth. The computer contains a decoding routine that automatically decodes the 65 and prints out an "A". Numbers can be represented in three ways inside the computer. The first is as a literal string using the ASCII representation in each number. No arithmetic can be done on numbers represented this way. The most common representation is called "floating point". The computer allocates a number of bytes or memory locations (usually five or six) to represent a number in a way which makes all the arithmetic possible. The last representation is called "Integer" where the range of the number is restricted to +/- 32767 as rounded integers. The only important part is that most micros handle data, either numeric or alphabetic, as the first classification- literal strings of ASCII characters. Since some arithmetic process is usually necessary, there are techniques for converting between the various data representations.

BASIC AND STRING HANDLING

A "string" is represented in BASIC by a "$" following the variable name. For example, A$, IN$, or X1$. A string is a series of ASCII coded characters stored together in a sequence in memory ("A$=ABCDE"). If we looked where the computer had stored A$, we would see 65,66,67,68,69 in adjacent memory locations. Note that in the example above the quotation marks indicate to the computer to store the ASCII values of the letters between them and are therefore mandatory.

Strings of ASCII characters can be manipulated by using a series of commands. The first are LEFT$, RIGHT$, and MID$. These commands allow us to view portions of strings and are used extensively in the retrieval and evaluation of data stored in the computer.

LEFT$
General form A$=LEFT$("ABCDE,2")
Print A$ will result in AB

The string to be manipulated is entered as a literal (as above in quotes) or as a variable (x$) followed by a comma and a number. The number designates how many characters starting at the left will be extracted from the main string. LEFT$ does not change and cannot be made to change the original string in any way.

RIGHT$
General form A$=RIGHT$("ABCDE",2)
Print A$ will result in DE

Obviously, RIGHT$ accomplishes the identical task as LEFT$ but the operation is performed on the other side.

MID$
General form A$=MID$("ABCDE,2,2)
Print A$ will result in BC

In this case, the first number following the string value is a pointer to the first character to be extracted, starting from the first character on the left (character A is the first, B the second, etc.). The next number designates the number of characters to be extracted. Note that MID$ cannot be used to modify the original string.

A$=MID$("ABCDE",2) this form of the MID$ command allows the user to remove the leftmost character ie
A$="CDE"

In addition to the above string handling commands there are another three instructions which allow manipulation or study of strings.

LEN
General form A=LEN("ABCDE")
Print A will result in 5

LEN returns the length of a string and is used extensively in setting up data in files. We will see more of LEN later.

VAL
General form A=VAL(124.35)
Print A will result in 124.35

VAL is used to convert literal numbers (as ASCII strings) to their "floating point" form so that arithmetic can be performed.

It is used widely for doing calculations on information stored in data files.

STR$
General form A$=STR$(A):WHERE A=124.35
Print LEN(A$) will result in 7

STR$ is the exact reverse of VAL. It is used to return calculated values to ASCII string form for storage in data files. An important note is that STR$ always leaves a space at the beginning of the string representation of the number for the sign (+ or-). If the number is positive it is preceded by a space or if it is negative by a minus sign.

The last two string handling commands are normally used in advanced programming to reduce storage requirements to the absolute minimum.

ASC(A$)
General form A=ASC("A")
Print A will result in 65.
This command converts a character to its ASCII code and allows it to be used in calculations.

CHR$(A)
General form A$=CHR$(65)
Print A$ will result in A
This is the reverse of ASC and uses the computer's coding logic to generate the appropriate character from a given code number.

STRING HANDLING OPERATORS

The following operators can be used alone and in combination with the above commands and information to allow very powerful decision making programs using string data. The operators are <,>,= and+.

Since strings are represented as numbers in the computer, they can be handled similarly in many respects. The ASCII code of both numbers ("2") and letters ascend with ascending values of alphabetic order. This allows us to compare apparent magnitudes of strings. It is true that the ASCII code of B(66) is greater than (>) A(65) so we can say that B$>A$ or A$<B$ of A$<>B$. All of these test would be true. This allows us to compare ASCII and place it in alphabetical or numerical order using simple comparison operators. The only difference in the use of these operators for numeric or string information is in the use of (+).

The (+) adds literal characters to the right hand end of the string and increases the string length (LEN) accordingly.

For example: A$="ABC"+"DE"
Print A will result in ABCDE

In this regard, it is totally different from its arithmetic counterpart.

The true power of the commands discussed up to this point can only be realised as they are used in combinations to compare and manipulate data.

PRINT USING

Anybody who has used the PET to output numeric data in tables will know how frustrating it can be to get the columns to align. A table with the following format is all too common:

No	Position	Time
0	1210153312	153.11123
10	12.312	16.215
17	.1273	5.12563127
5	150	0

There are quite a few one line routines which will chop a number around so that it will fit a set format, but none that I have seen which will allow true table layout.

I required a routine which would allow me to output a large amount of data into a table that would just fit an 80 column printer. The following program is the result. It works with disk files but it can easily be modified by changing the channel number so that it reads from tape. i.e. change 8 to 1 in line 270 and miss off the rest of the line:-

```
270 OPEN5,1
```

Lines 140-630 have to be altered to give the required data. The input routine shown is for an engine test bed running on petrol or methanol.

Lines 150-260 are the program constants some of which are sent to the printer.

Lines 280-360 are the variables that changed each time the engine was run.

Lines 370-450 perform calculations on the input data.

Lines 460-600 send the data to disk.

When modifying the program it is only necessary to ensure the following items:

1. The first item to be sent to disk for each data block is 1. Line 460 sends CH to disk: CH=1.

2. The last item to be sent before closing the disk file is 99999 as in line 630.

3. The title for the output data is held in F$.

4. The data format is held in P$ using 'I' to denote Integer output and 'F' to denote Floating point. Spaces act as delimiters between numbers.

5. Ensure that the number of items sent to disk for each line matches the number of format fields in P$!

Any number out of range is shown by '##' in the printout.

How the Program Works

Take a simple output format string to be:

P$="SIISSSFF.FFFSSSI"

Each set of data to be output will be stored on disk as a group of 3 preceded by a check digit and the data block will be terminated by 99999. Thus for example there are three sets of data to be output this will be stored as follows:

1 5 2.3716215 3 1 10.1 2021 9 1 69
95.6275 4 99999

740
The first item is read from disk. If it equals 99999 then there is no more data and the program terminates. The number is discarded after the test.

The position pointer CO is set to the start of P$.

750
The character under the pointer is assigned to V$. The pointers FB and FA (Float Before, Float After decimal point) are set to 0. In the above example V$=" ":CO=1.

770
A space is printed and the program jumps to 990 which will increment CO.

This is repeated until CO-3 V$='I'.

790
The string is now searched until a space is encountered or the end of the string is found.
810
FB is set to the number of I's encountered.

1100
A number is read from disk.

1030/1040
The number is turned into an integer NI and NI is turned into a string. If the length of N$ is greater than the number of characters set aside for it in FB or if N>IEIO an error message is printed and # symbols are sent to the printer.

1060
When a number is converted into a string it will have a preceding space if it is positive and a '-' if negative.

I have written the routine so that this space is removed for positive numbers thus saving space which is important on tightly packed output. Thus the user has to include an extra I or F into P$ when negative numbers are going to be output.

1070
If the number is negative then the - sign is put into the first position of the format. eg. N=-50

IIIII
\- 50

1080
If the number is positive then blanks are added. Thus numbers are always right justified.

The sample printout will now look like:

SSS

770
More spaces are encountered and sent to the printer. When the first 'F' is found CO=8.

840
The string is searched. FB is incremented when FA=0.

850
As soon as the decimal point is found then FA is incremented.

860
When FA>0, FA is incremented.

870
When the next space is found the program branches.

900
A Floating point number can be considered as an integer + a bit left over. Hence the number before the decimal point is sent to the printer using the integer subroutine in 1030. A decimal point is printed. CS is set to 0

910
As soon as a number drops below .01 it is output in exponential format. When this happens it is necessary to convert it back into floating format by adding in some zeros.

920
CS is set equal to the exponent. eg:

1.237 E- 05
N CS

The actual number is made to conform to the floating point remainder by dividing IO ie:

N=0.1237

950
If 0's are now added between the point and the next number it will be made to conform to true Floating point format. ie. 0.00001237.

970
If CS > FA then only 0's will be output. Starting from the decimal point in N numbers are printed. If the numeric output has still not filled the FA field then trailing '0''s will be printed to pad it out.

980
CO is incremented by the length of the format FB+FA

There are two operations that the program will not perform these being alphanumeric and exponential output which brings me to the competition.

Modify the program so that alphanumeric data can be input and formatted and add a routine which will convert any number into exponential format. Use A in P$ for alpha and E for exponential. I would suggest that a block of E's be used with a minimum of 8

being necessary to give output. ie:

EEEEEEEEEE
-1.763E-10

Set up the program so that it uses cassette Input/Output and will format to the screen (OPEN1,3) send your modified program to me, Dave Middleton, enclosing a SAE for return of your cassette. The closing date is 1.10.80 and the prize is any program from the Master Library.

```
 10 REM******NOTE -  ALL REM STATEMENTS
 11 REM*********MAY BE OMMITED**********
 20 REM*************START**************
 95 REM*********F$ IS TITLE STRING******
 96 REM**********P$ IS FORMAT STRING*****
 100 F$="RUN SP   N  LOAD     PH     MA     MF    AFR  Y.VOL · TQ     BP     YB  SFC"
 110 F$=F$+"     BMEP"
 120 P$=" II II  F.F FF.FF FF.FF FF.FF FF.FF FF.FF FFF.F FF.FF FF.FF FF.FF F.FF"
 130 P$=P$+"FF FFF.FF"
 132 REM
 135 REM******INPUT MAIN VARIABLES******
 140 OPEN10,4
 150 INPUT"DATE";DA$:PRINT#10,"DATE          ";DA$
 160 INPUT"JET SIZE";JE$:PRINT#10,"JET SIZE    ";JE$
 170 INPUT"NEEDLE TYPE";NT$:PRINT#10,"NEEDLE TYPE ";NT$
 180 IFTY$="M"THENPRINT#10,"FUEL TYPE: METHANOL"
 190 IFTY$="P"THENPRINT#10,"FUEL TYPE: PETROL"
 200 CH=1:INPUT"ATOMSPHERIC PRESSURE";PA
 210 PRINT#10,"ATMOSPHERIC PRESSURE";PA
 220 INPUT"FUEL TYPE METHENOL OR PETROL";TY$:IFTY$="M"THENCV=30000:FD=700
 230 PRINT#10,"FUEL TYPE: ";:IFTY$="P"THENPRINT#10,"PETROL"
 240 IFTY$="P"THENCV=43900:FD=753
 250 IFTY$="M"THENPRINT#10,"METHENOL"
 260 PRINT"CALORIFIC VALUE";CV:PRINT"FUEL DENSITY     ";FD :CLOSE10
 264 REM
 265 REM***********INPUT DATA************
 270 OPEN5,8,9,"@1:STEF,S,W"
 280 INPUT"RUN NUMBER        ";C0
 290 INPUT"PISTON HEIGHT MM";PH
 300 INPUT"SPEED  1000RPM  ";N
 310 INPUT"LOAD        LBS  ";L
 320 INPUT"DELTA H  MMH20  ";H
 330 INPUT"TEMP          C  ";T
 340 INPUT"VOLUME       ML  ";V
 350 INPUT"FUEL TIME  SEC  ";TS
 360 INPUT"SPARK TIME  BTC ";SP
 364 REM
 365 REM******PERFORM CALCULATIONS******
 370 MA=4*SQR((H*PA)/(T+273))
 380 MF=(FD*V)/TS/1000
 390 AF=MA/MF
 400 YV=51.78*(SQR((H*(T+273))/PA))/N
 410 TQ=1.446*L
 420 BP=0.1514*L*N
 430 YB=(BP*100000)/(MF*CV)
 440 SF=8.2/YB
 450 BM=9.116*L
 454 REM
 455 REM*******PRINT DATA TO DISK*******
 460 PRINT#5,CH:PRINT#5,C0:PRINT#5,SP:PRINT#5,N:PRINT#5,L:PRINT#5,PH:PRINT#5,MA
 470 PRINT#5,MF:PRINT#5,AF:PRINT#5,YV:PRINT#5,TQ:PRINT#5,BP:PRINT#5,YB
 590 PRINT#5,SF:PRINT#5,BM
 595 REM
 596 REM*********MORE DATA ?************
 610 INPUT"ANY MORE DATA Y/ ";AN$
 620 IFAN$<>"N"GOTO280
 630 PRINT#5,99999:CLOSE5
 644 REM
 645 REM******START OF PRINT USING******
 650 OPEN10,4:OPEN4,8,3,"1:STEF,S,R"
 654 REM
 655 REM*********PRINT TITLES************
 660 FORA=1TO80:PRINT#10,"*";:NEXT:PRINT#10
 670 PRINT#10,F$
 680 FORA=1TO80:PRINT#10,"*";:NEXT:PRINT#10
 693 REM
 694 REM*******MAIN PROG STARTS HERE***
 695 REM******TEST FOR END IE 99999*****
 740 CO=1:GOSUB1100:IFN=99999GOTO1010
```

```
744 REM
745 REM**START OF NEXT FORMAT FIELD**
746 REM*****TREAT SPACE AS AN END******
750 V$=MID$(P$,CO,1):FB=0:FA=0
760 IFV$=""GOTO990
770 IFV$=" "THENPRINT#10," ";:GOTO990
780 IFV$<>"I"GOTO830
784 REM
785 REM*********INTEGER ROUTINE********
790 FORC1=COTOLEN(P$):IFMID$(P$,C1,1)=" "GOTO810
800 NEXTC1:FB=LEN(P$)-CO+1:GOTO820
810 FB=C1-CO
820 GOSUB1100:GOSUB1030:CO=CO+FB-1:GOTO990
823 REM
825 REM*****FLOATING POINT ROUTINE*****
830 IFV$<>"F"GOTO63999
840 FORC1=COTOLEN(P$):D$=MID$(P$,C1,1):IFD$="F"ANDFA=0THENFB=FB+1
850 IFD$="."THENFA=1
860 IFD$="F"ANDFA>0THENFA=FA+1
870 IFD$=" "GOTO900
880 NEXTC1:IFFA>0THENFA=FA-1
900 GOSUB1100:GOSUB1030:PRINT#10,".";:CS=0:S=N
910 IFABS(N)>1E-2GOTO940
920 CS=INT(VAL(RIGHT$(STR$(N),2))):N=VAL(LEFT$(STR$(N),LEN(STR$(N))-2))/10
930 IFCS>0THENCS=CS-1
940 IFCS=>FATHENCS=FA-1
950 PRINT#10,LEFT$("000000000",CS);
960 V=2:IFABS(N)<1THENV=1
970 PRINT#10,MID$(STR$(N)+"00000000",LEN(STR$(INT(N)))+V,FA-CS-1);
980 CO=CO+FB+FA-1
985 REM
986 REM*****FINISH FORMAT ON THIS*****
987 REM******NUMBER. INCREMENT CO******
990 CO=CO+1:IFCO=>LEN(P$)THENPRINT#10:GOTO740
1000 GOTO750
1005 FORA=1TO80:PRINT#10,"*";:NEXT:PRINT#10
1010 PRINT#10:CLOSE10:CLOSE2
1020 STOP
1027 REM
1028 REM***TURN NUMBER INTO AN INTEGER
1029 REM*PRINT ERROR IF GREATER THAN FB
1030 NI=INT(N):N$=STR$(NI):IFLEN(N$)<FB+2ANDNI<1E9GOTO1060
1040 PRINT"ERROR NUMBER OUT OF RANGE";N:N$=LEFT$("##########",FB):GOTO1090
1060 N$=RIGHT$(N$,LEN(N$)-1)
1070 IFSGN(N)=-1THENN$=LEFT$("-         ",FB-LEN(N$))+N$:GOTO1090
1080 N$=LEFT$("          ",FB-LEN(N$))+N$
1090 PRINT#10,N$;:RETURN
1097 REM
1098 REM****READ A NUMERIC STRING FROM
1099 REMDISK CHOP OF CHR$(10) AT START*
1100 INPUT#4,CH$:CH$=MID$(CH$,2,200):N=VAL(CH$):PRINTN     :RETURN
READY.
```

```
DATE         30 JULY 1980
JET SIZE     1.001
NEEDLE TYPE 1510
ATMOSPHERIC PRESSURE 761
FUEL TYPE: METHENOL
********************************************************************************
RUN SP   N  LOAD    PH     MA     MF    AFR  V.VOL    TQ    BP    YB  SFC     BMEP
********************************************************************************
  1  0  1.0 25.00 25.00 51.10 70.00  0.73 263.4 36.15  3.78  0.18 #.4953 227.9
  2  0  2.0 29.00 29.00 36.83 22.55  1.63 101.2 41.93  8.78  1.29 6.3188 264.3
  3  5  3.0 60.00 50.00 41.82 14.00  2.98  82.5 86.76 27.25  6.48 1.2637 546.9

********************************************************************************
```

It's The Mu-pet Show!

OFFICIALLY APPROVED BY COMMODORE

Multi-User PET (Mu-pet) links 3-8 PET computers to one Commodore disc drive and a printer.

Mu-pet is very good news indeed for those PET users wanting a multi-user computer system and who, up until now, have run up against a budgetary brick wall.

Mu-pet delivers the goods at very low cost... which is one of the reasons it's become the world's biggest selling multi-PET system. Precisely engineered in the U.S. and Canada, Mu-pet makes the most of PET computers – *without the need for program changes.*

£595 is all it costs for a standard Mu-pet system that links three PET computers to a single Commodore disc drive and a printer. The cost of linking more PET computers, up to a maximum of eight, is £125 for each addition.

All machines have access to the disc drive and printer. The hardware which all runs via the IEEE bus has been so well designed that each PET thinks the disc is its own, and priority depends on who gets there first.

If you've three or more PETS, then you need a Mu-pet to make the most of them.

Yes, I want to see the Mu-pet show – please advise me on my nearest dealer.

Name____________________

Address____________________

KOBRA MICROSYSTEMS
14 The Broadway
West Ealing London W13 0SR
01-579 5845

KOBRA

CPUCN

The Use of WAIT

The command WAIT can most usefully be used to test to see if a particular bit is set in a byte.

Implementing the command:-

The keyboard is buffered allowing up to 10 characters to be stored prior to processing. The number of keystrokes to be processed is stored in location 158 (525 Old ROM).

WAIT 158,1 will cause the current program to stop until a key is pressed. What happens is that current value of location 158 will be ANDed bit by bit with 00000001 until a match is made in one bit, then processing continues. This WAIT 158,1 will have the same effect as
10 GETA$:IFA$=""GOTO10.

To explain further here are two examples:

1. WAIT 158,8 (bit pattern 00001000). Each time a key is pressed location 158 will be incremented. No bit in 158 will match 00001000 until the key has been pressed eight times.

2. WAIT 158,7 (bit pattern 00000111) Will a key have to be pressed seven times before the programs continues? No. The first key pressed will have a bit pattern of 00000001 and this will make a bit match. Hence the program will continue.

A second argument may be added to the WAIT. e.g. WAIT59410,4,4
This will WAIT until the space key has been pressed. If you take a PEEK at 59410 you will find that it contains 255 until a key on the same line as the STOP key is pressed (RVS 1 space < . =). The space key causes 251 (bit pattern 11111011) to appear in 59410.

It is not possible to use WAIT as any number other than 4 (0000100) will cause the WAIT to be fulfilled and 4 will cause the processor to wait indefinitely.

The first argument performs AND with location 59410.

```
        251 11111011
AND       4 00000100
            --------
            00000000
```

This result is then Exclusively ORed (EOR) with the second argument.

An Exclusive OR is the same as OR (01 OR 10 = 11) except that 1 only appears in the result when there is a 0 in the location and a 1 in the argument for the bit.

An Exclusive OR has the opposite effect to OR in that a bit is only set in the result when there is a zero in the location and a corresponding 1 in the argument. e.g.:

```
    10101010      11110000      11111111
EOR 00001111  EOR 00001111  EOR 10101010
    --------      --------      --------
    00000101      00001111      00000000
```

```
OR normally has this effect 10101010
                         OR 01010101
                            --------
                            11111111
```

Going back to the result from WAIT 59410,4,4

```
     251  11111011
AND    4  00000100
          --------
          00000000
EOR    4  00000100
          --------
          00000100
```

And the program will continue.

CHANGING ARRAY NAMES

Mike Gross-Niklaus

1. The problem.

Over the months, in CPUCN and other publications, many useful subroutines have been listed and explained. Some of these operate on the elements of an array.

But what if you want to use the same subroutine for several arrays? Most of us set up the subroutine with a 'working array', copy the contents of the array to be processed to the working array, perform the subroutine and then copy the processed working array back to the original one. Take a look at diagram 1 to see what I'm getting at.

The trouble is that if the array is a long one, this transferring can take a noticeable time. An alternative approach is to find the reference to the particular array in the array table, change the name to that of the 'working array', perform the subroutine, then change the name back again.

2. Steps involved in changing the name.

First define the old and new names and convert them into the same form as held in the array table.

Then hunt through the table for the old name.

Finally change the name by poking the new code.

To change the array name back again involves the same process, reversing the order of the defined old and new names.

3. Defining the names.

The technique shown here will only work on arrays of the same type (ie real, integer

or string array.) So, assuming that you know what type of array the subroutine is going to process, only the alpha numeric parts of the array name need be specified. For example, whether you want to change XX$() to WK$() or XX() to WK(), the procedure will be the same.

The method used requires you to set up in a parameter string (A$), the old and new names as two characters each, using a space if the name is one character only. For example, to change XX$ to Z$ requires A$="XX,Z ".

This parameter string is then passed to a subroutine, 40900, which dissects it to produce the codes for each of the two bytes used by the BASIC interpreter for array names. The rules for these are set out in appendix A of the User Manual. Briefly, the ASCII codes of the alpha-numeric characters of the array name are used, with zero in the second byte if the name has only one character. To these values are added 128 in the second byte if the array is not a real number (floating point) array, and 128 to the first byte if it is an integer array. In the example shown, real arrays are assumed, so the only processing of the ASCII codes required is to replace CHR$(32), (space) by a zero.

4. Hunting for the coded name.

The interpreter maintains some useful pointers in the 'scratch pad' (zero page) section of memory which you can get at with your BASIC programs by using PEEK.

Relevent here are the pointers for the start and end of arrays. The start is held as two bytes in locations 44 (126 old ROM) and 45 (127), and the end in locations 46 (128) and 47 (129). To convert these references into decimal, the second byte is multiplied by 256 and added to the first byte. For example if the first byte contains a PEEKed value of 4 and the second a value of 9, the decimal location is 9 times 256 plus 4 = 2308.

Using these calculations, you can determine the start and end of the search area.

The arrays are laid out with the first two btes as the coded name and the third and forth defining how many bytes to the next array name. So the technique is to check the name pointed at by locations 44 and 45, then if there is no match, add the value of the next two bytes to the pointer and so on, until you find a match or you reach the end of the array area. This skipping search technique is exactly that employed by the interpreter every time it comes across a reference to an array element in your BASIC program, i.e. it searches from the start of the array table each time.

[Incidentally, the same technique is used by the interpreter for looking up normal variables and even line numbers during GOTOs and GOSUBS. Program operation speeds can be significantly improved by putting your most used variables, arrays and subroutines 'at the front of the queue!!']

5. Example program.

The example program shows the use of a Bubble sort subroutine on two real number arrays XX and YY. Only 20 elements have been considered to allow listing of the unsorted and sorted arrays on the screen without scrolling or the need for screen page techniques.

The program fills the two arrays with random integer values in the range 1 to 99, and displays them for you to peruse. (I.e "Nothing up my sleeve!!"). After you press the space bar, each array in turn has it's name changed to Z, the bubble sort is done on array Z, then the name is changed back again. Finally the sorted arrays are displayed with the oportunity for another run in case you can't believe your eyes!

Incidently I lay out most of my programs in the manner shown if there is sufficient space. (qv this month's TEACHIN column.)

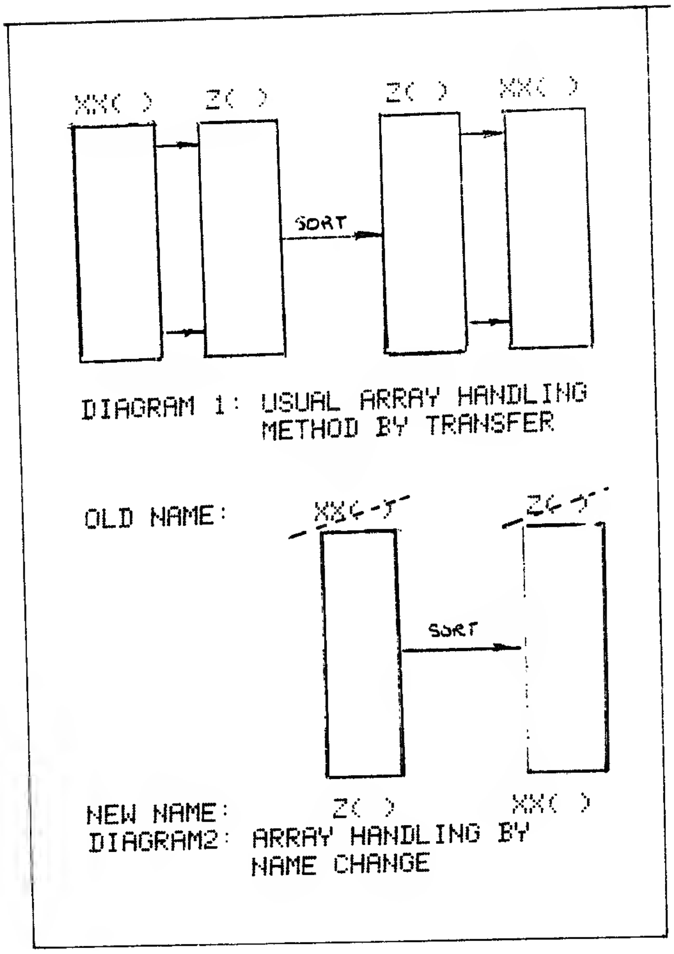
6. Conclusions.

You may feel that it's a lot of work for a not very big reward! Well that depends on the size of your arrays and how frequently you process them during the run of the program. As with all subroutines, once they are written and proven, they are available for instant use whenever you feel the need. In the case of one program I wrote to control industrial equipment, the increase in speed was essential. Using the normal transfer techniques would have caused the system to fail. The program was in PROM so I couldn't POKE the array name change into the BASIC text, which is messy anyhow if you are likely to amend the program with consequent changes in the relevent POKE locations. I could have reverted to machine code, but this method was far easier to implement, and fast enough.

I hope you get some mileage out of the idea, either by using it as intended, or just as an encouragement to you to have a look around the Scratch Pad locations and see what you can do by PEEKing and POKEing.

Mike Gross-Niklaus

```
10 REM TITLE:- ARRAY NAME CHANGE
20 REM BY    :- MIKE GROSS NIKLAUS
30 REM FOR   :- CPCUN
40 REM DATE  :- 02/07/80
99 REM ==========
100 REM DEMONSTRATION ONLY
999 REM ==========
1000 REM PRELIMINARIES
1001 REM ----------
1010 DIM XX(20), YY(20): REM TWO ARRAYS
1020 A=RND (-TI): REM RANDOMISE
1030 PT=0: EA=0
1099 REM ==========
1100 REM FILL ARRAYS WITH RNDM NOS
1101 REM ----------
1110 DEF FNR (X)=INT (RND (1)*X+1)
1120 FOR I = 1 TO 20
1130 XX(I)=FNR(99): YY(I)=FNR(99)
1140 NEXT I
1199 REM ==========
1200 REM CONFIRM ORIGINAL SEQUENCE
1201 REM ----------
1210 GOSUB41000
1220 PRINT"PRESS SPACE BAR TO CONTINUE"
1230 Z$=" ":GOSUB50000
1299 REM ==========
2000 REM SORT ARRAY XX
2001 REM ----------
2010 A$="XX,Z ": GOSUB40000: IF EF = 1 THEN END
2020 T$="XX": GOSUB50800
2030 A$="Z ,XX": GOSUB40000
2099 REM ==========
3000 REM SORT ARRAY YY
3001 REM ----------
3010 A$="YY,Z ": GOSUB40000: IF EF = 1 THEN END
3020 T$="YY": GOSUB50800
3030 A$="Z ,YY": GOSUB40000
3099 REM ==========
4000 REM CONFIRM SORTED ARRAYS
4001 REM ----------
4010 GOSUB 41000
4099 REM ==========
5000 REM AGAIN?
5001 REM ----------
5010 PRINT"AGAIN (Y OR N)?
5020 Z$="YN":GOSUB50000:IF I=1 THEN RUN
```



```
5030 END
5099 REM ==========
40000 REM CODE OLD ARRAY NAME
40001 REM ----------
40010 X$=LEFT$ (A$,2): GOSUB 40900
40099 REM ==========
40100 REM FIND OLD ARRAY NAME
40101 REM ----------
40110 EF=0:REM ERROR FLAG RESET
40120 PT=PEEK(44)+PEEK(45)*256
40130 EA=PEEK(46)+PEEK(47)*256
40140 IF PEEK (PT)  <>NC(1) THEN 40160
40150 IF PEEK (PT+1)= NC(2) THEN 40200
40160 PT=PT+PEEK(PT+2)+PEEK(PT+3)*256
40170 IF PT<EA THEN 40140
40180 EF=1: REM ERROR FLAG
40190 RETURN
40199 REM ==========
40200 REM CHANGE THE NAME
40201 REM ----------
40210 X$= RIGHT$(A$,2):GOSUB 40900
40220 FORI = 0 TO 1 : POKE PT+I, NC(I+1): NEXT I
40230 RETURN
40299 REM ==========
40900 REM CODE THE ARRAY NAME
40901 REM ----------
40910 FOR I = 1 TO 2: NC(I)=ASC (MID$ (X$,I,1)): NEXT I
40920 IF NC(2)=32 THEN NC(2)=0
40930 RETURN
40999 REM ==========
41000 REM CONFIRM ARRAYS
41001 REM ----------
41010 PRINT "SEQUENCE","ARRAY XX","ARRAY YY"
41020 FORI = 1 TO 20
41030 PRINT RIGHT$ (STR$ (I),2), XX(I), YY(I)
41040 NEXT I
41050 RETURN
41099 REM ==========
50000 REM AWAIT VALID KEY
50001 REM ----------
50010 GET A$: IF A$="" THEN 50010
50020 FOR I = 1 TO LEN (Z$)
50030 IF A$=MID$ (Z$,I,1) THEN RETURN
50040 NEXT I: GOTO 50010
50090 REM ==========
50800 REM BUBBLE SORT
```

```
50801 REM ----------
50810 PRINT"□■SORTING ARRAY ";T$
50820 X=19: REM X  IS CURRENT BOTTOM
50830 SI=0: REM SI IS SWOP INDICATOR
50840 PRINT"■■■SORTING COUNTDOWN";X;"■ "    :FOR I = 1 TO X
50850 IF Z(I)<=Z(I+1)THEN 50870
50860 A=Z(I): Z(I)=Z(I+1): Z(I+1)=A:SI=1
50870 NEXT I
50880 IF SI=1 THEN X=X-1: GOTO 50830
50890 RETURN
50899 REM ==========

READY.
```

Shell-Metzner Sort

W Murcott BSc MBCS

In "Printout" in issue 7, Mike Gross-Niklaus mentions the Shell-Metzner sort. This has been around a long time. I first came across it in the mid 1960s, in an Algol work-book from English Electric at Kidsgrove. The acknowledgement given in the work-book says that it was first described by D. L. Shell (Comm. A.C.M. 2 No 7 (1959)). The beauty of this sort, in comparison with many others, is that it is quite "intelligent" in that its speed increases if the raw data has already a degree of order.

```
59000 REM*****************SHELL SORT INCLUDES ELEMENT 0*************************
59002 REM
59005 REM***********0 TO A IS ARRAY SIZE (A+1 ELEMENTS)*************************
59008 N=A: M=A+1
59010 M=INT(M/2): IF M=0 THEN RETURN
59020 J=0: K=N-M
59025 IF J>K GOTO 59010
59030 I=J
59035 IF I<0 GOTO 59055
59050 IF A$(I+M)<A$(I) GOTO 59060
59055 J=J+1: GOTO 59025
59060 F$=A$(I): A$(I)=A$(I+M): A$(I+M)=F$: I=I-M: GOTO59035
READY.
```

Trace

A G Price
Principal Lecturer Mathematics
Liverpool Polytechnic

"PET users may be interested in my experience with Brett Butler's TRACE routine, published in CPUCN Volume 2 Issue 3. As written, tracing can be slowed down by use of the POKE instruction indicated. This introduces a delay of about 300 milliseconds multiplied by the value POKEd, so that the built-in value of 3 causes execution at about 1 second per line. The delay for a value of 1 is rather variable, and is not recommended. A value of 0 corresponds to 256, and gives a delay of about 80 seconds. The delay is not interruptable by use of either the SHIFT or STOP keys, which can be a little frustrating.

By rearranging the TRACE program slightly, it can be made to respond to the SHIFT key at any time, thus giving the effect of a 'single-shot' key by setting a large delay and tapping the SHIFT key to advance to the next instruction. Hold it down for full-speed running. To STOP the program whilst in the delayed state, press RUN-STOP and SHIFT together.

There is a short-cut in the program which can cause the TRACE to miss the start of a new line. It occurs if a GOTO is performed from a line number 256*K+J (where J is between 0 and 255 and K is any integer) to line 256*K, e.g. line 260 to line 256, or line 1300 to line 1280.

Users will have noticed that, when TRACE displays a READ statement, the characters

read appear in the displayed line immediately following the READ.

*   *   *   *   *   *   *   *   *

A one-liner which may be found useful in some applications is the following expression, which yields the number of elapsed days since 1st January 1900 for a date expressed in the form D = day (1-28,29,30 or 31), M = month (1-12)m Y = year (0-99). It produces correct results for all dates from March 1st 1900 onwards (to 29th February 2100, to be precise):-

```
D+INT(365.25*(Y+(M<3))+0.1)+INT(30.6*(M-12
*(M<3))-31.35)
```

Modifications to TRACE to test SHIFT continuously

Changes are underlined.

```
104 DATA 253,208,4,228,254,240,106,133,253,
     133,35,134,254,134,36,169

120 DATA 3,133,107,165,152,208,10,202,
     208,249,136,208,246,198,107,208

136 DATA 242,32,-54,169,160,160,80,153,
     255,127,136,208,250,132,182,132

860 DATA 228,78,240,107,133,77,133,82,134,
     78,134,83,169,3,133,74,173,4

870 DATA 2,208,10,202,208,248,136,208,
     245,198,74,16,241,32,-54,169,160

1120 PRINT "CHANGE SPEED WITH:
      POKE";S1+119;" ,X"
```

NOTE: The modifications have been tested on an upgraded ROM machine (model 3032) but not an original machine.

Hex-Dec/Dec-Hex

The following routine was sent to us by R W Brand. It allows conversion from HEX to DEC or DEC to HEX. Press RETURN without an input to jump to the opposite conversion direction.

The program is a bit long ......maybe somebody can get it down to 6 lines or so......

```
5 REM*****************DECIMAL TO HEXADECIMAL AND VICE-VERSA**********************
10 CLR:B=0:C=-1:INPUT"HEX███*███";A$:IFA$="*"GOTO120
20 FORA=LEN(A$)TO1STEP-1:B$=MID$(A$,A,1)
30 IFB$="A"THENB$="10"
40 IFB$="B"THENB$="11"
50 IFB$="C"THENB$="12"
60 IFB$="D"THENB$="13"
70 IFB$="E"THENB$="14"
80 IFB$="F"THENB$="15"
90 C=C+1:B=B+((16↑C)*VAL(B$))
100 NEXT
110 PRINT"DECIMAL IS "B"█":GOTO10
120 CLR:N=16:INPUT"DECIMAL███0███";A:IFA=0GOTO10
130 FORC=1TO4:A$(C)="0":NEXT
140 B=B+1
150 C=INT(A/N):A$(B)=STR$((A/N-C)*N):A=C:GOSUB200
160 IFA<NTHENB=B+1:A$(B)=STR$(A):GOSUB200:GOTO180
170 GOTO140
180 A$=A$(4)+A$(3)+A$(2)+A$(1):PRINT"HEX IS ";A$"█"
190 GOTO120
200 IFA$(B)=" 10"THENA$(B)="A":RETURN
210 IFA$(B)=" 11"THENA$(B)="B":RETURN
220 IFA$(B)=" 12"THENA$(B)="C":RETURN
230 IFA$(B)=" 13"THENA$(B)="D":RETURN
240 IFA$(B)=" 14"THENA$(B)="E":RETURN
250 IFA$(B)=" 15"THENA$(B)="F":RETURN
260 A$(B)=RIGHT$(A$(B),1):RETURN
READY.
```

BASIC REPEAT KEY

This little program gives you the chance of making a very simple repeat key for the PET.

Location 151 (515 old ROM) is used to hold the last key pressed. This location is the keyboard matrix number and while it can be used for fairly simple applications such as cursor control, it is not realistic to use it for ASCII conversion within a program.

The PET uses a ROM based look up table for converting the key matrix into ASCII. This table starts at 59127 (E6F7 hex). By adding on the key matrix number an ASCII character will be returned.

When the shift key is being pressed location 152 (516) goes from 0 to 1. When this is multiplied by 128 you will obviously get 0 or 128. By ORing this with the result from the PEEK at the lookup table the shifted ASCII characters are produced.

The routine will not work for old ROM PETs due to PEEK protect, but it should be very simple to write a machine code program to perform the above operations and store the result somewhere that can be PEEKed by BASIC.

```
 10 REM**********************************************
 15 REM
 20 REM CURSOR REPEAT - PAUL HIGGINBOTTOM - COMMODORE
 25 REM
 26 REM USE CURSOR CONTROL KEYS TO MOVE *
 27 REM
 30 REM**********************************************
 40 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶*▌▌";
 50 A=(PEEK(59127+PEEK(151))OR(PEEK(152)*128)
 55 IFA=253GOTO50
 60 IFA=17THENPRINT" ▒▌▌*▌▌";
 70 IFA=145THENPRINT" ▒▌*▌▌";
 80 IFA=157THENPRINT" ▌▌*▌▌";
 90 IFA=29THENPRINT" *▌▌";
 100 GOTO50
READY.
```

Applications

A R Clark C D Smith I Kirk
Leeds University Physics Dept

FAR INFRARED ASTRONOMY GROUND STATION
Using the PET with Interrupts

The schematic overleaf shows the arrangement of a Ground Station to be used later this year in Texas during high-altitude balloon flights.

The slow scan video gives us the star field within an 8 x 10 field of view and the star field is updated every 2 seconds.

The 6800 synchronises to the serial data stream and decodes 32 analog words and 8 digital words. Two of the digital words give us our infrared information and the M6800 performs a handshake of 128 bytes of I.R. data every 8 seconds to the PET. (Each handshake taking approximately 100 milliseconds).

Most of the other digital words define the status of our experiment and appropriate statements are displayed on the Kode VDU - the status being upgraded every 4 seconds.

The infrared data is displayed graphically on the PET screen, partially processed and then stored on the Floppy. As the printer takes approximately 15 seconds to print out a full page, we can obtain a hard copy every 3 handshakes.

The following notes give further details of our technique for handshaking data between the PET and external microprocessor.

PET - M6800 DATA HANDSHAKE

Figure 1 shows the basic flow diagram for the handshaking of 128 bytes of data from a Motorola M6800 into the PET. The data arrives from the M6800 onto the user port data lines (memory location $E841), and the interrupt request comes to the memory expansion port which is also connected to a floppy disk. The handshaking routine is designed so that this operation is carried out before the PET services any of its internally-generated interrupts, and resides in the cassette #2 buffer - starting address $033A (826 decimal). The machine code routine is shown in Figure 2.

In order that the above routine is executed before any internal interrupts, the vector pointer must be reset so that PET jumps directly to $033A. This could be done directly by the commands: POKE 537,(low order byte): POKE 538,(high order byte), but if an interrupt occurs before the high order byte has been stored, the system will crash. For this reason, the pointer must be reset by the machine code routine shown in Figure 3, which is loaded from BASIC into the top of cassette #2 buffer, and run by a SYS command. If PET is servicing a routine when the M6800 sends IRQ, it will continue with that routine, but IRQ will still be low when it finishes, so PET will service M6800 immediately on completion of the RTI instruction. Hence the M6800 will continue to request until PET acknowledges by sending the 'DATA REQUEST' pulse.

(The handshake routine finishes by storing a flag in $0377 [887 decimal] which can be tested from BASIC by the PEEK(887) command, always remembering to reset the flag as soon as it has been found.)

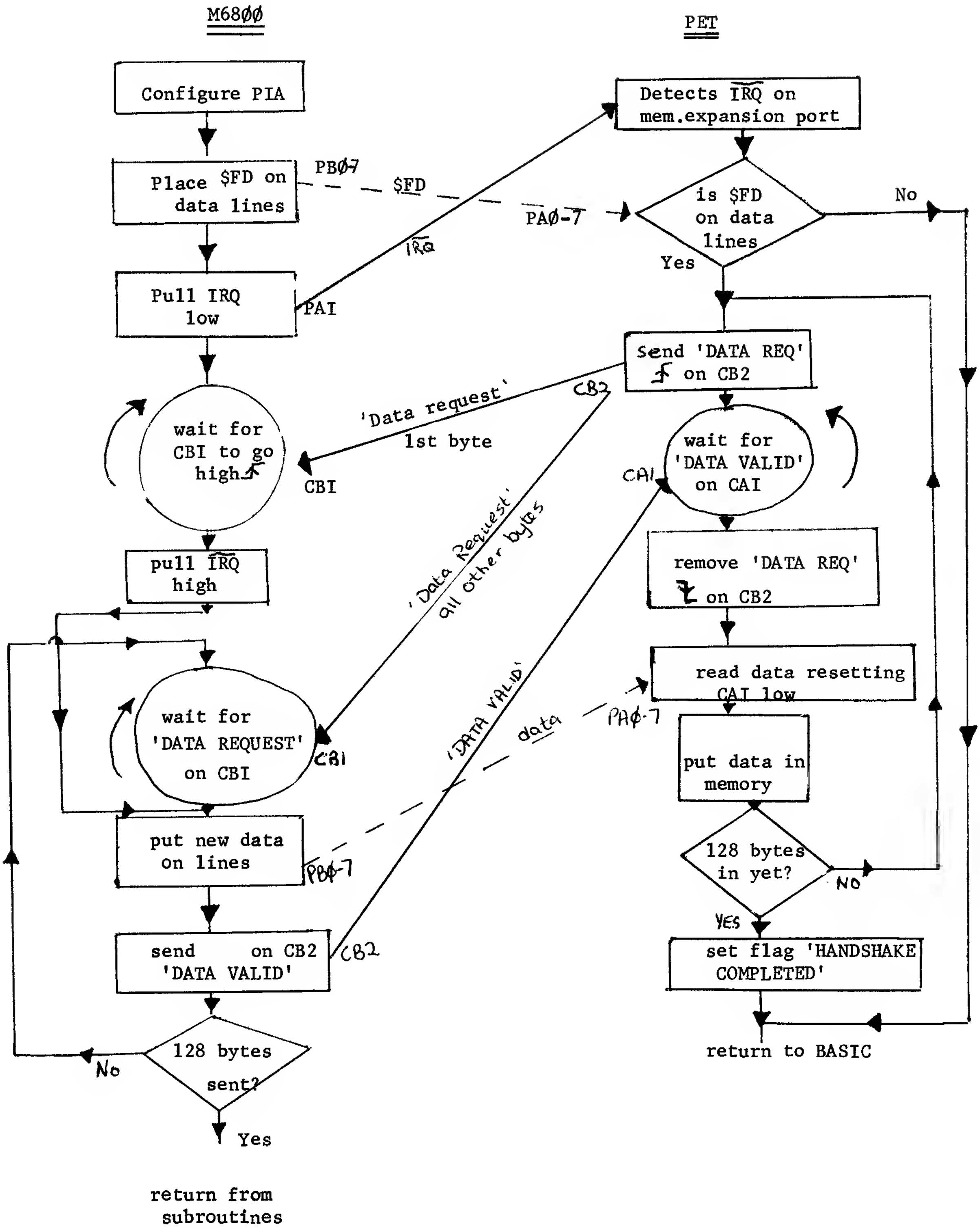


FIGURE 1 Flow chart for data handshaking

```
        LDA     $E841       ;READ DATA
        CMP     #$FD        ;IS $FD ON THE DATA LINES?
        BNE     EXIT
        LDY     #0
L2      LDA     #$E1        ;$FD ON THE LINES -
        ORA     $E84C       ;  AND 'DATA REQUEST'
        STA     $E84C
L3      LDA     #2
        AND     $E84D       ;WAIT FOR 'DATA VALID' ON CAI
        BEQ     L3
        LDA     #$DF
```

```
AND    $E84C      ;PULL CB2 - REMOVE 'DATA REQUEST'
STA    $E84C
LDA    $E841      ;PUT DATA IN STORE
STA    $0378,Y    ;RESETTING CA1
INY
CPY    #$80
BNE    L2         ;DONE?
LDA    #$FF
STA    $0377      ;SET FLAG 'HANDSHAKE COMPLETED'
                  ;TO BE TESTED BY BASIC
JMP    BASIC      ;RETURN TO BASIC
```

Figure 2 PET's machine code programme for handshaking

```
4 REM*************************************************************************
5 REM PET MACHINE CODE PROGRAMME FOR HANDSHAKING
6 REM*************************************************************************
10 DATA 173,65,232,201,253,208,41,160,0,169,225,13,76,232,141,76,232,169,2
20 DATA 45,77,232,240,249,169,232,45,76,232,141,76,232,173,65,232,153,120,3,200
30 DATA 192,128,208,222,169,255,141,119,3,76,133,230
97 REM*************************************************************************
98 REM MACHINE CODE PROGRAMME TO MOVE PET INTERRUPT POINTER TO 826 (033A HEX)
99 REM*************************************************************************
100 DATA 120,169,58,141,25,2,169,3,141,26,2,88,96
110 REM************************************************************************
READY.
```

```
SEI
LDA   (LOW ORDER BYTE)
STA   $0219
LDA   (HIGH ORDER BYTE)
STA   $021A
CLI
RTS
```

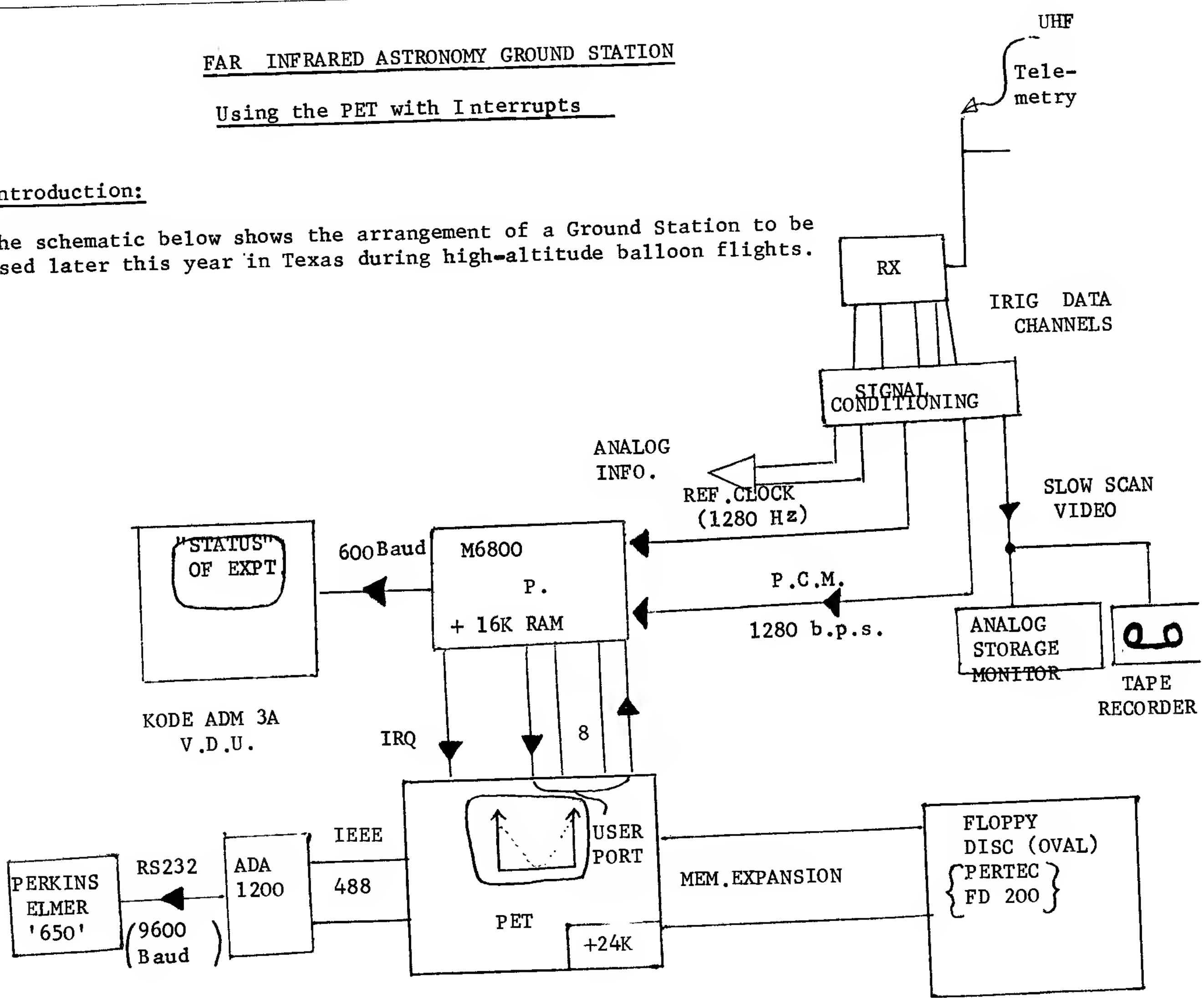
Figure 3 Machine code program to reset PET's interrupt pointer

FAR INFRARED ASTRONOMY GROUND STATION

Using the PET with Interrupts

Introduction:

The schematic below shows the arrangement of a Ground Station to be used later this year in Texas during high-altitude balloon flights.



Paul Higginbottom

INSTRUCTION MODES AND 6502 OPERATIONS

You remember last time I left you a program that I said I would let you ponder over, and that I would go over it this time ? Well here goes:

The program demonstrated many different principles of 6502 machine code programming, and instruction modes. These will be explained statement by statement.

When I say "instruction mode", this means that there are different modes for different instructions, and, in fact, many instructions can be executed in more than one mode. An example of his is the LDA instruction, used last time. It can be used in the "immediate" mode. This means that the accumulator can be loaded directly with a value, rather than the contents of a memory location. Another mode of the LDA instruction is "absolute". This means that the address is given after the instruction (instead of a value), and the Accumulator is loaded with the contents of the memory at the specified address.

line 1: LDA #$01 - This is an immediate mode instruction and 'says' :load the accumulator with the value $01 (the '#' symbol tells the assembler that is a value rather than the contents of memory location $01 that is to be loaded into the accumulator). Immediate mode means that the value to be used is immediately after the instruction, rather than the contents of a memory location defined by the operand.

line 2: LDX #$00 - This is also an immediate mode instruction and 'says' :load the X register with the value $00.

line 3: STA $8000,X - This is an absolute indexed mode instruction and says: store the contents of the accumulator at the address [$8000 + the contents of the X register]

line 4: INX - This is an implied mode instruction since the instruction itself defines what register the operation is being performed on. The instruction says:- Increment the X register by one.

line 5: CPX #$28 - This is an immediate mode instruction and says: compare the contents of the X register with $28

line 6: BNE $033E - This is an implied mode instruction and says: branch if not equal to the address $033E. Thus, if the comparison of X and $28 (or decimal 40) was not equal, i.e X<>40, then branch back to the address $033E, and continue executing. The BRANCH instructions will be explained fully later in this article.

line 7: RTS - This is an implied mode instruction and says: return from subroutine. This instruction will cause execution of this routine to cease, and return to where it was executed from. As this was a SYS call, then it will return to a usual basic 'ready' mode.

In the processor, there are many registers, some I have already mentioned. I think that now would be a good time to describe those registers.

ACCUMULATOR - This is the main eight bit register, around which most of the instructions are based.

X register - This is an eight bit index register. 'Index' means that it can be used to offset a base address by its contents. This was shown in the first program.

Y register - This is also an eight bit index register and has similar facilities as the X register. However some 6502 instructions are designed only for use with the X or the Y registers but not both.

Status register - This is possibly the most valuable and vital register in the processor. It uses each of its eight bits to describe a state that has arisen by one instruction, or by a sequence of instructions.

Each bit will be described below:-

N V B D I Z C

N - Negative Flag.

This is set if a negative condition is made. This could be done with a subtraction, decrement, comparison or some other instruction. There are no instructions to set it directly. (The comparison instruction, does an internal subtraction and sets the flags according to the result of the subtraction). BMI (branch on minus) will take the branch if the negative flag is set.

V - Overflow Flag.

This is set if a result actually overflowed the register being operated on, e.g if a value is added to a register that makes it go over $FF (or 255 decimal), then the value in the register would be the result-256, and this might make mathematical calculations go wrong, and so the overflow flag is set. It may be cleared directly with the instruction CLV (clear overflow flag). It may be detected by the instructions BVC (branch on overflow clear) and BVS (branch if overflow set).

B - BRK command executed.

This is set if the BRK (break) instruction is executed. It would not be worthwile covering this at the moment.

D - Decimal mode flag.

This is set and cleared by SED and CLD (set, and clear decimal mode) instructions. If decimal mode is set, then the add and

subtract instructions work in a different way. The results are left in BCD (binary coded decimal).

Decimal mode off.

3   9 (base sixteen - HEX)

0011 1001 = 57 in decimal.

Decimal mode on.

5   7 (base ten - decimal)

0101 0111 = 57 in decimal

As can be seen, all that happens is that the result is transformed to leave the first decimal digit in the first four bits, and the second, the upper four bits. This is useful for decimal output, rather than binary or hex.

I - Interrupts disabled flag.

This is set if the SEI (set interrupt disabled flag) is executed, and reset if the CLI (clear interrupt disabled flag) instruction is executed. It would not be worthwile covering this at the moment.

Z - Zero flag.

This is set when any instruction leaves a register with 0 in it. It cannot be set or reset directly. It can be detected with the BEQ (branch if equal), and BNE (branch if not equal) instructions.

C - Carry flag.

This can be thought of as a ninth bit to any register. It can be manipulated directly with the SEC (set carry) and the CLC (clear carry) instructions. It can be detected by the BCC (branch if carry clear) and BCS (branch if carry set) instructions. If 2 two byte (16 bit) values are to be added, and stored in a result, then it is necessary to detect a carry from the first addition of the two low bytes, which should be added to the result of the addition of the two high bytes. An example of this would be as follows:-

Imagine we have two 16 bit values stored in the locations FRED, and BERT. This value is of course split into two 8 bit values. So we have the first value in FRED and FRED+1, and the second in BERT and BERT+1. To add these two values and store the result in JOE and JOE+1, the following code is necessary:-

```
    CLC          ;CLEAR CARRY FLAG TO
                 ;START WITH
    LDA FRED     ;LOAD THE ACCUMULATOR
                 ;WITH THE CONTENTS OF
                 ;FRED
                 ;THIS IS OF COURSE ONLY
                 ;THE FIRST 8 BITS OF
    ADC BERT     ;ADD WITH CARRY THE
                 ;CONTENTS OF BERT
    STA JOE      ;AND STORE THE RESULT
                 ;IN JOE
    LDA FRED+1   ;LOAD THE UPPER 8 BITS
                 ;OF FRED
    ADC BERT+1   ;ADD WITH THE CARRY THE
                 ;UPPER 8 BITS OF BERT
    STA JOE+1    ;AND STORE THE RESULT IN
                 ;THE UPPER 8 BITS OF JOE
```

At this point it becomes necessary to define what is available to us within the 6502 microprocessor, and its storage. The 6502 uses 2 bytes for each address apart from those in the range $0000 to $00FF (256) bytes, which do not need a high byte to describe their address, because in that range it is always zero. As was stated in the last article, each block of 256 bytes is known as a page of memory. The first 256 bytes ($0000 - $00FF) are known as page zero locations and there is a mode of instruction called simply 'zero page'. This mode tells the processor that it only need pick up one byte for the address.

The 6502 also has what is known as a 'stack', which is used by the processor as well as the programmer. This uses the whole of page one memory (that is $0100 - $01FF). I will try to explain simply what the stack is used for. When doing a GOSUB instruction in BASIC, this tells the Pet to jump to a given line number and when a RETURN instruction is found it must return to the statement following the GOSUB instruction. So, at the time the GOSUB instruction was encountered, it must have some way of remembering where it currently is in memory, so that it knows where to RETURN to. There are instructions PHA (push accumulator on stack), PHP (push processor status register on the stack), PLA (pull accumulator off stack), and PLP (pull processor status off stack), which work with an eight bit register inside the processor called a stack pointer. Each time a PHA instruction is done, the contents of the accumulator will be put at $0100 offset by the stack pointer, and the stack pointer will be automatically deceremented by one, so that the next push will put the data onto the next position on the stack etc. Correspondingly, the PLA instruction will load the accumulator with the contents of the memory whose address is $0100 offset by the stack pointer, and the stack pointer will be incremented by one etc.
N.B The movements of the stack pointer also apply with the PHP and PLP commands. Going back to our GOSUB instruction, by using the instructions just described, the PET may use the stack to save its current position when going off to the subroutine. When a RETURN instruction is encountered, it simply 'pulls' off the stack the point at which it was before entering the subroutine and moves onto the next instruction as if nothing had happened. This explains why there is a maximum level to 'nesting' of subroutines (i.e one subroutine calling another, which in turn calls another etc.), because the stack is of a fixed size and as each set of return data is pushed onto the stack, the stack slowly gets filled up, until an ?OUT OF MEMORY ERROR occurs. You can demonstrate this to yourself by typing in the basic program

```
10 C=C+1:GOSUB10 <RETURN>
RUN <RETURN>
```

The pet should (after a little while) display ?OUT OF MEMORY ERROR IN 10. If you print the value of C, this will tell you the number of levels of subroutine you had got up to before the stack was filled. The pet does in fact check to see if the stack will be over filled by the next instruction. If it didn't, then the pet would probably hang itself up. Fortunately MICROSOFT (the authors of the basic interpreter in the pet) thought of that. Right then, we've discussed the stack, and how useful that is, I think I should explain the BRANCH instructions fully now, and I will leave another program for you to look at.

BEQ - Branch if equal, meaning branch if zero flag set.
BNE - Branch if not equal, meaning branch if zero flag not set.
BCC - Branch if carry clear.
BCS - Branch if carry set.
BVC - Branch if overflow clear.
BVS - Branch if overflow set.
BMI - Branch if negative result, branch if negative flag is set
BPL - Branch if positive result, branch if negative flag is not set

Program to rotate the screen left one byte.

We shall write this one in standard assembler format.

```
 * =$033A          ;PROGRAM STARTS AT
                   ;$033A

   LDA #$00        ;LOAD ACCUMULATOR WITH
                   ;THE VALUE '$00'.
   STA $01         ;STORE IT IN MEMORY
                   ;LOCATION $01
   LDA #$80        ;LOAD ACCUMULATOR WITH
                   ;THE VALUE '$80'.
   STA $02         ;STORE IT IN MEMORY
                   ;LOCATION $02
```

(HERE I HAVE SET UP IN LOCATIONS 1 AND 2 THE 16 BIT ADDRESS OF THE START OF THE SCREEN - $8000. BYTES 1 AND 2 WILL BE USED AS A 16 BIT POINTER TO THE CURRENT LOCATION ON THE SCREEN BEING MOVED)

```
   LDA #$19        ;LOAD ACCUMULATOR WITH
                   ;THE VALUE 25 (DECIMAL)
   STA $00         ;STORE IT IN MEMORY
                   ;LOCATION $00
```

(HERE I HAVE SET UP IN LOCATION 0 A SCREEN LINE COUNTER. I HAVE SET IT TO 25 AND EACH TIME I ROTATE A ROW OF THE SCREEN, I WILL DECREMENT THIS COUNTER, AND STOP WHEN IT REACHES ZERO)

```
START  LDY #0          ;ZEROISE OFFSET TO
                       ;POINTER
                       ;IN 1 AND 2
       LDA ($01),Y     ;LOAD THE ACCUMULATOR
                       ;WITH
                       ;THE CONTENTS OF THE
                       ;ADDRESS
                       ;POINTED TO BY THE
                       ;BYTES $01
                       ;AND THE NEXT ONE
                       ;($02)
                       ;OFFSET BY THE
                       ;CONTENTS OF THE
                       ;Y REGISTER
       PHA             ;PUSH THE CONTENTS OF
                       ;THE ACCUMULATOR ONTO
                       ;THE STACK.  SINCE
                       ;THIS IS THE FIRST
                       ;CHARACTER IN THE ROW,
                       ;WE WILL SAVE IT. THEN
                       ;PULL THE OTHER 39
                       ;CHARACTERS BACK ONE
                       ;POSITION, AND PULL
                       ;THIS VALUE
                       ;BACK OFF THE STACK,
                       ;AND PUT IT AT THE
                       ;END OF THE ROW
       INY             ;INCREMENT THE OFFSET
                       ;BY ONE
LOOP   LDA ($01),Y     ;GET A CHARACTER FROM
                       ;THE ROW
       DEY             ;DECREMENT OFFSET TO
                       ;STORE THIS CHARACTER
                       ;IN THE PREVIOUS
                       ;POSITION
       STA ($01),Y     ;PUT IT BACK ON THE
                       ;SCREEN
       INY             ;INCREMENT Y TO WHAT
                       ;IT WAS
       INY             ;MOVE ONTO THE NEXT
                       ;SQUARE
       CPY #$28        ;HAVE WE REACHED THE
                       ;LAST POSITION ON THE
                       ;LINE ? I.E HAS THE
                       ;OFFSET
                       ;GOT TO 40 (DECIMAL)
       BNE LOOP        ;IF NOT - GO BACK AND
                       ;DO IT AGAIN
       PLA             ;WE HAVE - SO PUT
                       ;SAVED CHARACTER
                       ;BACK ONTO THE SCREEN
                       ;PULL BYTE OFF STACK
                       ;INTO THE ACCUMULATOR
       DEY             ;RESET OFFSET TO LAST
                       ;POSITION
       STA ($01),Y     ;PUT IT IN THE LAST
                       ;POSITION
       DEC $00         ;DECREMENT LINE COUNT
       BEQ EXIT        ;IF EQUAL TO ZERO -
                       ;THEN EXIT
       CLC             ;CLEAR CARRY
       LDA $01         ;OTHERWISE - BUMP
                       ;POINTER BY 40
       ADC #$28        ;ADD $28 TO LOW BYTE
       STA $01         ;AND STORE RESULT
       LDA $02         ;GET HIGH BYTE
       ADC #$00        ;ADD NOTHING + THE
                       ;CARRY
       JMP START       ;JUMP BACK TO THE
                       ;START

EXIT   RTS             ;FINISHED - RETURN
                       ;FROM SUBROUTINE
```

There you go. Have a look at this until next newsletter, when I will explain how it works.

Bye.

SUPERMON - OLD ROM VERSION

David A Hills

Here is a chance for those of you with the OLD ROM 8K PET to have a go with the superb SUPERMON program. The TIM monitor has been included in with SUPERMON so once it's been entered you should be able to put TIM into honourable retirement. All of the commands for SUPERMON are the same, as given in CPUCN Vol. 2 Issue 3.

Save SUPERMON using the TIM SAVE command from 16F6 to 2000.

To enter the program when it has been loaded type: SYS 6726.

SUPERMON

Modified for use with Old ROMs by :-

D.A. Hills,

19, St. Anthonys Drive,

Chelmsford,

Essex.

Monitor is entered by SYS(6726)

Exit to BASIC by .X

Save on tape with :-

.S 01,SUPERMON(0),16F6,2000

```
.: 16F0 AA AA AA AA AA AA A9 06
.: 16F8 8D 1B 02 A9 17 8D 1C 02
.: 1700 A9 43 85 21 D0 12 A9 42
.: 1708 85 21 D8 4A 68 85 1E 68
.: 1710 85 1D 68 85 1C 68 85 1B
.: 1718 68 69 FF 85 19 66 69 FF
.: 1720 85 1A BA 86 1F 58 20 BC
.: 1728 17 A6 21 A9 2A 20 EC 18
.: 1730 A9 52 85 0D D0 1B A9 00
.: 1738 85 CA 85 0D 85 0A 20 BC
.: 1740 17 A9 2E 20 D2 FF 20 5A
.: 1748 19 C9 2E F0 F9 C9 20 F0
.: 1750 F5 A2 07 DD CC 17 D0 0F
.: 1758 A5 20 85 0E 86 20 BD D4
.: 1760 17 48 BD DC 17 48 60 CA
.: 1768 10 E9 4C 5D 1A 38 A5 13
.: 1770 E5 11 85 0B A5 14 E5 12
.: 1778 A8 05 0B 60 A5 11 85 19
.: 1780 A5 12 85 1A 60 85 21 A0
.: 1788 00 20 04 19 B1 11 20 DD
.: 1790 18 20 C1 17 C6 21 D0 F1
.: 1798 60 20 28 19 90 0D A2 00
.: 17A0 81 11 C1 11 F0 05 68 68
.: 17A8 4C 65 17 20 C1 17 C6 21
.: 17B0 60 A9 1B 85 11 A9 00 85
.: 17B8 12 A9 05 60 A9 0D 4C D2
.: 17C0 FF E6 11 D0 06 E6 12 D0
.: 17C8 02 E6 0A 60 3A 3B 52 4D
.: 17D0 47 58 4C 53 18 18 17 18
.: 17D8 18 18 19 19 8B 7B F6 28
.: 17E0 A1 C7 68 68 20 50 43 20
.: 17E8 20 53 52 20 41 43 20 58
.: 17F0 52 20 59 52 20 53 50 A5
.: 17F8 0D D0 06 20 BC 17 20 01
.: 1800 19 20 01 19 A2 00 BD E4
.: 1808 17 20 D2 FF E8 E0 13 D0
.: 1810 F5 20 BC 17 A2 2E A9 3B
.: 1818 20 EC 18 20 01 19 20 D2
.: 1820 18 20 B1 17 20 85 17 F0
.: 1828 4D 20 5A 19 20 19 19 90
.: 1830 48 20 09 19 20 5A 19 20
.: 1838 19 19 90 3D 20 09 19 A0
.: 1840 00 B9 14 1A 30 06 20 D2
.: 1848 FF C8 D0 F5 29 7F 20 D2
.: 1850 FF 20 2A F3 F0 20 A6 0A
.: 1858 D0 1C 20 6D 17 90 17 20
.: 1860 BC 17 A2 2E A9 3A 20 EC
.: 1868 18 20 01 19 20 CE 18 A9
.: 1870 08 20 85 17 F0 DB 4C 36
.: 1878 17 4C 65 17 20 28 19 20
.: 1880 19 19 90 03 20 7C 17 20
.: 1888 B1 17 D0 0A 20 28 19 20
.: 1890 19 19 90 E5 A9 08 85 21
.: 1898 20 5A 19 20 99 17 D0 F8
.: 18A0 F0 D4 20 CF FF C9 0D F0
.: 18A8 0C C9 20 D0 CC 20 19 19
.: 18B0 90 03 20 7C 17 A6 1F 9A
.: 18B8 A5 1A 48 A5 19 48 A5 1B
.: 18C0 48 A5 1C A6 1D A4 1E 40
.: 18C8 A6 1F 9A 4C 8B C3 A2 01
.: 18D0 D0 02 A2 09 B5 10 48 B5
.: 18D8 11 20 DD 18 68 48 4A 4A
.: 18E0 4A 4A 20 F5 18 AA 68 29
.: 18E8 0F 20 F5 18 48 8A 20 D2
.: 18F0 FF 68 4C D2 FF 18 69 06
.: 18F8 69 F0 90 02 69 06 69 3A
.: 1900 60 20 04 19 A9 20 4C D2
.: 1908 FF A2 02 B5 10 48 B5 12
.: 1910 95 10 68 95 12 CA D0 F3
.: 1918 60 20 28 19 90 02 85 12
.: 1920 20 28 19 90 02 85 11 60
.: 1928 A9 00 85 0F 20 5A 19 C9
.: 1930 20 D0 09 20 5A 19 C9 20
.: 1938 D0 0E 18 60 20 4F 19 0A
.: 1940 0A 0A 0A 85 0F 20 5A 19
.: 1948 20 4F 19 05 0F 38 60 C9
.: 1950 3A 08 29 0F 28 90 02 69
.: 1958 08 60 20 CF FF C9 0D D0
.: 1960 F8 68 68 4C 36 17 4C 65
.: 1968 17 20 5A 19 A9 00 85 EE
.: 1970 85 FA A9 23 85 F9 20 28
.: 1978 19 29 0F 85 F1 20 5A 19
.: 1980 A2 00 20 CF FF C9 2C F0
.: 1988 55 C9 0D F0 0B E0 10 F0
.: 1990 F1 95 23 E6 EE E8 D0 EA
.: 1998 A5 20 C9 06 D0 C8 A2 00
.: 19A0 8E 0B 02 A5 F1 D0 03 4C
.: 19A8 65 17 C9 03 B0 F9 20 67
.: 19B0 F6 20 3B F8 20 FF F3 A5
.: 19B8 EE F0 08 20 95 F4 D0 08
.: 19C0 4C 65 17 20 AE F5 F0 F8
.: 19C8 20 4D F6 20 22 F4 20 8A
.: 19D0 F8 20 13 F9 AD 0C 02 29
.: 19D8 10 D0 E5 4C 36 17 20 19
```

```
19E0 19 A5 11 85 F7 A5 12 85
19E8 F8 20 CF FF C9 20 F0 F9
19F0 C9 0D F0 A4 C9 2C F0 03
19F8 4C 66 19 20 19 19 A5 11
1A00 85 E5 A5 12 85 E6 A5 20
1A08 C9 06 F0 32 A2 00 20 B1
1A10 F6 4C 36 17 0D 20 20 20
1A18 20 20 20 20 20 20 20 30
1A20 20 20 31 20 20 32 20 20
1A28 33 20 20 34 20 20 35 20
1A30 20 36 20 20 B7 00 00 00
1A38 98 48 20 BC 17 68 A2 2E
1A40 20 EC 18 4C 01 19 AD FE
1A48 1F 35 36 AD FF 1F 85 87
1A50 4C F6 16 00 00 A9 3F 20
1A58 D2 FF 4C 36 17 A2 08 DD
1A60 DE 1F D0 0E 86 20 8A 0A
1A68 AA BD E9 1F 48 BD E3 1F
1A70 48 60 CA 10 EA 4C 55 1A
1A78 A2 02 2C A2 00 B4 11 D0
1A80 08 B4 12 D0 02 E6 0A D6
1A88 12 D6 11 60 20 5A 19 C9
1A90 20 F0 F9 60 A9 00 8D 0F
1A98 00 20 8C 1A 20 2F 19 20
1AA0 1C 19 90 09 60 20 5A 19
1AA8 20 19 19 B0 DE 4C 55 1A
1AB0 20 04 19 CA D0 FA 60 E6
1AB8 13 D0 02 E6 14 60 A2 02
1AC0 B5 10 48 BD 2B 00 95 10
1AC8 68 9D 2B 00 CA D0 F1 60
1AD0 AD 2C 00 AC 2D 00 4C DD
1AD8 1A A5 13 A4 14 38 E5 11
1AE0 85 EC 98 E5 12 A8 05 EC
1AE8 60 20 94 1A 20 09 19 20
1AF0 A5 1A 20 BE 1A 20 A5 1A
1AF8 20 D9 1A 20 09 19 90 15
1B00 A6 0A D0 64 20 D0 1A 90
1B08 5F A1 11 81 13 20 B7 1A
1B10 20 C1 17 D0 EB 20 D0 1A
1B18 18 A5 EC 65 13 85 13 98
1B20 65 14 85 14 20 BE 1A A6
1B28 0A D0 3D A1 11 81 13 20
1B30 D0 1A B0 34 20 78 1A 20
1B38 7B 1A 4C 27 1B 20 94 1A
1B40 20 09 19 20 A5 1A 20 09
1B48 19 20 5A 19 20 28 19 90
1B50 14 85 21 A6 0A D0 11 20
1B58 D9 1A 90 0C A5 21 81 11
1B60 20 C1 17 D0 EE 4C 55 1A
1B68 4C 36 17 20 94 1A 20 09
1B70 19 20 A5 1A 20 09 19 20
1B78 5A 19 A2 00 20 5A 19 C9
1B80 27 D0 14 20 5A 19 9D 31
1B88 00 E8 20 CF FF C9 0D F0
1B90 22 E0 20 D0 F1 F0 1C 8E
1B98 0F 00 20 2F 19 90 C6 9D
1BA0 31 00 E8 20 CF FF C9 0D
1BA8 F0 09 20 28 19 90 B6 E0
1BB0 20 D0 EC 86 20 20 BC 17
1BB8 A2 00 A0 00 B1 11 DD 31
1BC0 00 D0 0C C8 E8 E4 20 D0
1BC8 F3 20 CE 18 20 04 19 20
1BD0 C1 17 A6 0A D0 92 20 D9
1BD8 1A B0 DD 4C 36 17 20 94
1BE0 1A 8D 2E 00 A5 12 8D 2F
1BE8 00 A9 04 A2 00 85 25 86
1BF0 26 A9 93 20 D2 FF A9 16
1BF8 85 21 20 18 1C 20 6D 1C
1C00 65 11 84 12 C6 21 D0 F2
1C08 A9 91 20 D2 FF 4C 36 17
1C10 A0 2C 20 38 1A 20 CE 18
1C18 20 04 19 A2 00 A1 11 20
1C20 7C 1C 48 20 C2 1C 68 20
1C28 D8 1C A2 06 E0 03 D0 12
1C30 A4 23 F0 0E A5 FF C9 E8
1C38 B1 11 B0 1C 20 65 1C 88
1C40 D0 F2 06 FF 90 0E BD 4A
1C48 1F 20 4D 1D BD 50 1F F0
1C50 03 20 4D 1D CA D0 D5 60
1C58 20 70 1C AA E8 D0 01 C8
1C60 98 20 65 1C 8A 86 20 20
1C68 DD 18 A6 20 60 A5 23 38
1C70 A4 12 AA 10 01 88 65 11
1C78 90 01 C8 60 A8 4A 90 0B
1C80 4A B0 17 C9 22 F0 13 29
1C88 07 09 80 4A AA BD F9 1E
1C90 B0 04 4A 4A 4A 4A 29 0F
1C98 D0 04 A0 80 A9 00 AA BD
1CA0 3D 1F 85 FF 29 03 85 23
1CA8 98 29 8F AA 98 A0 03 E0
1CB0 8A F0 0B 4A 90 08 4A 4A
1CB8 09 20 88 D0 FA C8 88 D0
1CC0 F2 60 B1 11 20 65 1C A2
1CC8 01 20 B0 1A C4 23 C8 90
1CD0 F1 A2 03 C4 25 90 F2 60
1CD8 A8 B9 57 1F 8D 2C 00 B9
1CE0 97 1F 8D 2D 00 A9 00 A0
1CE8 05 0E 2D 00 2E 2C 00 2A
1CF0 88 D0 F6 69 3F 20 D2 FF
1CF8 CA D0 EA 4C 04 19 20 94
1D00 1A 20 C1 17 20 C1 17 20
1D08 09 19 20 A5 1A 20 09 19
1D10 20 01 19 20 D9 1A 90 09
1D18 98 D0 13 A5 EC 30 0F 10
1D20 07 C8 D0 0A A5 EC 10 06
1D28 20 DD 18 4C 36 17 4C 55
1D30 1A 20 94 1A A9 03 85 21
1D38 20 5A 19 20 99 17 D0 F8
1D40 AD 2E 00 85 11 AD 2F 00
1D48 85 12 4C F1 1B C5 26 F0
1D50 03 20 D2 FF 60 A9 03 A2
1D58 24 85 25 86 26 20 BC 17
1D60 78 A9 84 8D 19 02 A9 1D
1D68 8D 1A 02 A9 A0 8D 4E E8
1D70 CE 13 E8 A9 28 8D 48 E8
1D78 A9 00 8D 49 E8 AE 1F 00
1D80 9A 4C B8 18 20 FB FC 68
1D88 8D 1E 00 68 8D 1D 00 68
1D90 8D 1C 00 68 8D 1B 00 68
1D98 8D 19 00 68 8D 1A 00 BA
1DA0 8E 1F 00 58 20 BC 17 20
1DA8 B1 17 85 21 A0 00 20 8C
1DB0 17 20 04 19 AD 1A 00 85
1DB8 12 AD 19 00 85 11 20 CE
1DC0 18 20 18 1C 20 2A F3 C9
1DC8 F7 F0 F9 20 2A F3 D0 03
1DD0 4C 36 17 C9 FF F0 F4 4C
1DD8 60 1D 00 20 94 1A 20 09
1DE0 19 8E 32 00 A2 03 20 8C
1DE8 1A 48 CA D0 F9 A2 03 68
1DF0 38 E9 3F A0 05 4A 6E 32
1DF8 00 6E 31 00 88 D0 F6 CA
1E00 D0 ED A2 02 20 CF FF C9
1E08 0D F0 1E C9 20 F0 F5 20
1E10 F0 1E B0 0F 20 3C 19 A4
1E18 11 84 12 85 11 A9 30 9D
1E20 31 00 E8 9D 31 00 E8 D0
1E28 DB 8E 2C 00 A2 00 86 0A
1E30 A2 00 86 21 A5 0A 20 7C
1E38 1C A6 FF 8E 2D 00 AA BD
1E40 97 1F 20 D5 1E BD 57 1F
1E48 20 D5 1E A2 06 E0 03 D0
1E50 12 A4 23 F0 0E A5 FF C9
1E58 E8 A9 30 B0 1D 20 D2 1E
1E60 88 D0 F2 06 FF 90 0E BD
1E68 4A 1F 20 D5 1E BD 50 1F
1E70 F0 03 20 D5 1E CA D0 D5
1E78 F0 06 20 D2 1E 20 D2 1E
1E80 AD 2C 00 C5 21 D0 59 20
1E88 09 19 A4 23 F0 2B AD 2D
1E90 00 C9 9D D0 1C 20 D9 1A
1E98 90 09 98 D0 4A A6 EC 30
1EA0 46 10 07 C8 D0 41 A6 EC
```

```
.:  1EA8 10 3D CA CA 8A A4 23 D0
.:  1EB0 03 B9 12 00 91 11 88 D0
.:  1EB8 F8 A5 0A 91 11 20 6D 1C
.:  1EC0 85 11 84 12 A0 41 20 38
.:  1EC8 1A 20 CE 18 20 04 19 4C
.:  1ED0 DE 1D 20 D5 1E 86 20 A6
.:  1ED8 21 DD 31 00 F0 0C 68 63
.:  1EE0 E6 0A F0 03 4C 30 1E 4C
.:  1EE8 55 1A E8 86 21 A6 20 60
.:  1EF0 C9 30 90 03 C9 47 60 38
.:  1EF8 60 40 02 45 03 D0 08 40
.:  1F00 09 30 22 45 33 D0 08 40
.:  1F08 09 40 02 45 33 D0 08 40
.:  1F10 09 40 02 45 B3 D0 08 40
.:  1F18 09 00 22 44 33 D0 8C 44
.:  1F20 00 11 22 44 33 D0 8C 44
.:  1F28 9A 10 22 44 33 D0 08 40
.:  1F30 09 10 22 44 33 D0 08 40
.:  1F38 09 62 13 78 A9 00 21 81
.:  1F40 82 00 00 59 4D 91 92 86
.:  1F48 4A 85 9D 2C 29 2C 23 28
.:  1F50 24 59 00 58 24 24 00 1C
.:  1F58 8A 1C 23 5D 8B 1B A1 9D
.:  1F60 8A 1D 23 9D 8B 1D A1 00
.:  1F68 29 19 AE 69 A8 19 23 24
.:  1F70 53 1B 23 24 53 19 A1 00
.:  1F78 1A 5B 5B A5 69 24 24 AE
.:  1F80 AE A8 AD 29 00 7C 00 15
.:  1F88 9C 6D 9C A5 69 29 53 84
.:  1F90 13 34 11 A5 69 23 A0 D8
.:  1F98 62 5A 48 26 62 94 88 54
.:  1FA0 44 C8 54 68 44 E8 94 00
.:  1FA8 B4 08 84 74 B4 28 6E 74
.:  1FB0 F4 CC 4A 72 F2 A4 8A 00
.:  1FB8 AA A2 A2 74 74 74 72 44
.:  1FC0 68 B2 32 B2 00 22 00 1A
.:  1FC8 1A 26 26 72 72 88 C8 C4
.:  1FD0 CA 26 48 44 44 A2 C8 04
.:  1FD8 22 10 20 2D 2F 33 54 46
.:  1FE0 48 44 43 2C 41 49 4E 00
.:  1FE8 E8 1A 3C 1B 6A 1B DD 1B
.:  1FF0 FD 1C 30 1D DA 1D 54 1D
.:  1FF8 55 FD 84 1D 5D 1A F6 16
.?

READY.
```

EDUCATION NEWS

Regional Educational Conferences

We are pleased to announce that we will be putting on conferences for teachers in Glasgow, Manchester, Birmingham, Bristol and London in mid September.

Further details will be included in the next issue of CPUCN and the new addition of Microcomputers in Schools and Colleges.

Admission to the conference will be free.

Topics to be covered will include, Software development for the classroom and first hand accounts of experiences with the Pet. We would like to contact teachers who are using a Pet in these areas with a view to them contributing to the conference.

Teachers who would like to give a short paper at the conference, are asked to contact:

CBM Ltd.,
Nick Green,
360 Euston Road,
London,
W.1.

Free Workshop Software

If you are using Pets in your school, college or university, and teachers in your area are using these Pets, for example, out of school hours for software development purposes or training, we will send you on cassette tape some 43 programs from North America and Canada. The programs cover a wide range of ages and subject matters and some are written to a very high standard indeed. Others you will be less impressed with! These programs maybe modified in any way you wish and distribited in any way you wish. We hope that they will inspire you to new heights. In a few months time we hope to have collected from you improved versions and indeed new programs that you would like distributed in the same way.

Please send me free Workshop Software 1.
We are allowing teachers in our area access to Pets, when not in use, for Workshop purposes:

NAME: ______________________________

ADDRESS: ______________________________

NO. OF PETS: ______________________________

PERIPHERALS: ______________________________

AVAILABILITY OF PETS:

a) Dates ______________________________

b) Times ______________________________

Does your Institution offer courses to teachers?

If so, please give details below:

Would you be interested in attending a free one-day course on Pet Architecture in London for Workshop organizers?

Please fill in the above and send to:

Commodore,
Education Department,
360 Euston Road,
London,
W.1.

The PET/CMB computer has been around for over two years and, in that time, various changes have been made to the system software stored in the ROMs. To bring you up to date and clarify any confusion that may have been caused by the number of different ROM sets, we are publishing the following list with explanations.

When the PET 2001 first went into production, there were two ROM Sets incorporated into the system. One ROM Set is the 6540 type ROM. This is a 28 Pin Rom which is manufactured by MOS Technology, Inc. You will find these ROMs in the following locations on the PET 2001-4K and 2001-8K Main Logic Board:

Location	ROM	Part Number
H1	6540-019	901439-09 or 01
H2	6540-013	901439-02
H3	6540-015	901439-03
H4	6540-016	901439-04
H5	6540-012	901439-05
H6	6540-014	901439-06
H7	6540-018	901439-07
A2	6540-010	901439-08

NOTE: There is an 019 ROM at the H1 location. On some earlier Main Logic Boards you will find a 6540-011 at H1. This ROM has been updated to an 019 due to an intermittent bug in the edit software. This ROM Set is Basic level 1.

The other ROM Set incorporated into the PET 2001 is a type 2316B 24 Pin ROM. You will find these ROMs in the following locations on the PET 2001-4K and 2001-8K Main Logic Board:

Location	ROM	Part Number
H1	901447-09	901447-09
H2	901447-03	901447-03
H3	901447-05	901447-05
H4	901447-06	901447-06
H5	901447-02	901447-02
H6	901447-04	901447-04
H7	901447-07	901447-07
A2	901447-08	901447-08

NOTE: There is an 09 ROM at the H1 location. On some earlier Main Logic Boards you will find a 901447-01 ROM. This ROM has been updated to an 09 ROM due to an intermittent bug in the edit software. Like the 6540 ROM Set, this too is a Basic Lever 1 ROM Set. To determine what the 6540 and 2316B ROMs listed above are capable of, I would refer you to the "PET User Manual" Model 2001-8.

The next two ROM Sets are Basic Level II ROMs, and are fitted as standard on all 16 and 32K PET/CBMs. They are also Retrofit Kits for the 2316B and 6540 Basic Level 1 ROMs. The Basic Level II ROMs include the machine language. Basic Level II allows you to interface the Commodore 2040 Dual Floppy to your PET/CBM. Basic Level I ROMs will not allow you to interface the 2040 Dual Floppy to your PET. The Basic Level II Retrofit ROMs also allow you to use arrays with more than 255 elements.

If your PET/CBM has the Basic Level I 6540 ROMs, you could use the following ROMs which come in the form of a Retrofit Kit to upgrade your PET/CBM to Basic Level II.

Location	ROM	Part Number
H1	6540-020	901439-13
H2	6540-022	901439-15
H3	6540-024	901439-17
H4	6540-025	901439-18
H5	6540-021	901439-14
H6	6540-023	901439-16
H7	6540-026	901439-19

If your PET/CBM has the Basic Level I 2316B ROMs, you would use the following ROMs which come in the form of a Retrofit Kit to upgrade your PET/CBM to Basic Level II:

Location	ROM	Part Number
H1	901465-01	901465-01
H2	901465-02	901465-02
H3	901465-24	901465-24
H4	901465-03	901465-03
H5	Blank	
H6	Blank	
H7	Blank	

The following ROM Sets are the ROMs that are currently being used in production. There are two sets of ROMs in use. If you have a graphic style PET, you should have the following ROMs in your unit:

Location	ROM	Part Number
D3	Blank	
D4	Blank	
D5	Blank	
D6	901465-01	901465-01
D7	901465-02	901465-02
D8	901447-24	901447-24
D9	901465-03	901465-03
F10	901447-10	901447-10

If your computer is a business style, you should have the following ROMs in your unit:

Location	ROM	Part Number
D3	Blank	
D4	Blank	
D5	Blank	
D6	901465-01	901465-01
D7	901465-02	901465-02
D8	901447-01	901447-01
D9	901465-03	901465-03

The ROMs in the graphic and business PET/CBM are Basic Level III ROMs. To determine what any machine fitted with Basic Level III is capable of, you should refer to the "CBM User Manual" Model 2001-16, 16N, 32, 32N.

The ROMs currently being used in production of the 3040 Dual Floppy are as follows:

Location	ROM	Part Number
UL1	901468-06	901468-06
UK1	Blank	
UH1	901468-07	901468-07
UK3	6530-02	901466-02

These ROMs are DOS Version I.

New 80 column PET BASIC 4.0 DOS 2.1 Retrofit ROMs will be available in about 60 days time at £38.00 BASIC 4.0, £38.00 dos2.1.

New BASIC eliminates garbage collection problems and has a powerful direct access feature which considerably enhances the disk system.

BASIC has been extended to include a number of D.U.M. (Disk Utility Maintenance) features, considerably improving "User Friendliness". DOS 2.1 simply enables the new commands to be recognised.

NEW ROM TO OLD ROM CONVERSION TABLES FOR PAGE ZERO

NEW PET DEC	NEW PET HEX	OLD PET DEC	OLD PET HEX	FUNCTIONS
0	$0000-$0002		$0000-$0002	POKE LOCATION CHANGE
1-2	$0000-$0002	1-2	$0000-$0002	USER FUNCTION ADDRESS LO;HI
3.	$0003	90	$005A	GENERAL COUNTER FOR BASIC SEARCH [SEARCH CHRTR-USUALLY ':' OR ENDLIN]
4.	$0004	91	$005B	;00 USED AS DELIMITER [SCAN BETWEEN QUOTES FLAG]
5.	$0005	92	$005C	GENERAL COUNTER FOR BASIC [INPUT BUFFER PNTR][# OF SUBSCRIPTS]
6.	$0006	93	$005D	FLAG TO REMEMBER DIMENSIONED VARIABLES [1ST CHAR OF ARRAY NAME]
7.	$0007	94	$005E	FLAG FOR VARIABLE TYPE 0=NUMERIC;1=STRING [$FF STRING]
8.	$0008	95	$005F	FLAG FOR INTEGER TAPE [80=INTEGER; 00=FLOATING POINT
9.	$0009	96	$0060	FLAG TO CRUNCH RESERVED WORDS [DATA SCAN FLAG/LIST QUOTE FLAG]
10.	$000A	97	$0061	FLAG WHICH ALLOWS SUBSCRIPTS IN SYNTAX [FN X FLAG]
11.	$000B	98	$0062	FLAG INPUT OR READ [0=INPUT;64=GET;152=READ]
12.	$000C	99	$0063	FLAG SIGN OF TAN [FLAG FOR TRIG SIGNS/ COMPARSN EVALUATION FLAG]
13.	$000D	100	$0064	FLAG TO SUPPRESS OUTPUT [+=NORMAL; -=SUPPRESSED]
14.	$000E	3	$0003	ACTIVE I/O CHANNEL [PROMPT-SUPPRESS]
15.	.....	6	.....	TERMINAL WIDTH (UNUSED)
16.	.....	7	.....	LIMIT FOR SCANNING SOURCE [COLUMNS UNUSED]
17-18	$0011-$0012	8-9	$0008-$0009	LINE NUMBER BEFORE STORAGE [INTEGER ADDRESS FROM BASIC
19.	$0013	101	$0065	INDEX TO NEXT AVAILABLE DESCRIPTOR [VARIBL DESCRPTR STACK PNTR]
20-21	$0014-$0015	102-103	$0066-$0067	POINTER TO LAST STRING TEMPORARY LO;HI [SECOND DESCRPT PONTER]
22-29	$0016-001E	104-111	$0068-$0070	TABLE OF DOUBLE BYTE DESCRIPTORS WHICH POINT TO VARIABLES
23				[DESCRIPTOR STACK FOR TEMPORARY STRINGS]
30-31	$001F-0020	112-113	$0071-$0072	INDIRECT ADDRESS #1 LO;HI [POINTER FOR NUMBER TRANSFER]
32-33	$0021-$0022	114-115	$0073-$0074	INDIRECT INDEX #2 LO;HI [NUMBER POINTER]
34-39	$0023-$0027	116-121	$0075-$0078	PSEUDO REGISTER FOR FUNCTION OPERANDS
40-41	$0028-$0029	122-123	$007A-$007B	POINTER TO START OF BASIC TEXT AREA LO;HI
42-43	$002A-$002B	124-125	$007C-$007D	POINTER TO START OF VARIABLES LO;HI [END OF BASIC/START VARBL]
44-45	$002C-$002D	126-127	$007E-$007F	POINTER TO ARRAY TABLE LO;HI [END VARIABLES/ START ARRAYS]
46-47	$002E-$002F	128-129	$0080-$0081	POINTER TO END OF VARIABLES LO;HI [START OF AVAILBL SPACE PTR]
48-49	$0030-$0031	130-131	$0082-$0083	START OF STRINGS POINTER LO;HI [BOTTOM OF STRINGS (MOVING DOWN)]
50-51	$0032-$0033	132-133	$0084-$0085	TOP OF STRING SPACE POINTER LO;HI [MOVING DOWN]
52-53	$0034-$0035	134-135	$0086-$0087	HIGHEST RAM ADDRESS LO;HI [TOP OF BASIC MEMORY]
54-55	$0036-$0037	136-137	$0088-$0089	CURRENT LINE BEING EXECUTED (54=2 MEANS DIRECT)[CURRENT LN #]
56-57	$0038-$0039	138-139	$008A-$008B	LINE NUMBER FOR CONTINUE COMMAND LO;HI [LINE # SAVED BY END]
58-59	$003A-$003B	140-141	$008C-$008D	NEXT STATEMENT TO EXECUTE LO;HI [PREV LINE# FOR CONT]

60-61	$003C-$003D	142-143	$008E-$008F	DATA LINE# FOR ERRORS LO;HI [LINE# OF DATA LINE]
62-63	$003E-$003F	144-145	$0090-$0091	DATA STATEMENT POINTER LO;HI [READ POINTER]
63......................................				[MEMORY ADDRESS OF DATA LINE]
64-65	$0040-$0041	146-147	$0092-$0093	SOURCE OF INPUT LO;HI [INPUT VECTOR (DATA ETC)][DATA STMT PTR]
66-67	$0042-$0043	148-149	$0094-$0095	CURRENT VARIABLE NAME [CURRENT VARIABLE SYMBOLS]
68-69	$0044-$0045	150-151	$0096-$0097	POINTER TO VARIABLE IN MEMORY LO;HI [CURRENT VARBLE START ADR]
70-71	$0046-$0047	152-153	$0098-$0099	POINTER TO VARIABLE REFERRED TO IN CURRENT FOR-NEXT
72-73	$0048	154-155	$009A	POINTER TO CURRENT OPERATOR IN TABLE LO;HI
73......................................				[154 Y SAVE REGISTER/NEW OPERATOR SAVE]
74.	$004A	156	$009C	SPECIAL MASK FOR CURRENT OPERATOR [CMPR SYMBL ACMLTR <1=2>4]
75-76	$004B-$004C	157-158	$009D-$00A1	FUNCTION DEFINITION POINTER LO;HI [NUMBER WORK AREA FOR SQR..]
77-78	$004D-$0050	159-160	$009D-$00A1	POINTER TO A STRING DESCRIPTION LO;HI [NMBR WRK AREA 157-161]
79.	$004D-0050	161	$009D-$00A1	LENGTH OF ABOVE STRING
80.	$004D-$0050	162	$00A2	CONSTANT USED BY GARBAGE COLLECT ROUTINE [3 OR 7 FOR GRBG CLT]
81.	$0051-$0053	163	$00A3-$00A5	$4C CONSTANT-6502 JMP INSTRUCTION [JUMP VECTOR FOR FUNCTIONS]
82-83	$0051-$0053	164-165	$00A3-$00A5	VECTOR FOR FUNCTION DISPATCH LO;HI
84-89	$0054-$0058	166-171	$00A6-$00AA	FLOATING ACCUMULATOR #3 [NUMERIC STORE AREA]
90-91	$0059-$005D	172-173	$00AB-$00AF	BLOCK TRANSFER POINTER #1 LO;HI [NUMERIC STORE AREA]
92-93	$0059-$005D	174-175	$00AB-$00AF	BLOCK TRANSFER POINTER #2 LO;HI [NUMERIC STORE AREA]
94-99	$005E-$0063	178-181	$00B0-$00B5	FLOATING ACCUMULATOR #1 (USER FUNCTIIION EVALUATED HERE)
95......................................				[PRIMARY ACCUMLATR E,M,M,M,M,S][MSB PARAMETERS 181-FLPT SIGN]
100	$0064	182	$00B6	DUPLICATE COPY OF MANTISSA OF FAC #1 [TAYLOR SERIES CONST CNTR]
101	$0065	183	$00B7	COUNTER - NUMBER OF BITS TO SHIFT TO NORMALIZE FAC #1 [SEE 103]
102-107	$0066-$006B	184-189	$00B8-$00BD	FLOATING ACCUMULATOR #2 [SECONDRY ACUMULTR HLDNG AREA]
103......................................				[101 IS ACCUMULATOR HIGH ORDER PROPOGATION WORD]
108	$006C	190	$00BE	OVERFLOW BYTE FOR FLOATING ARGUMENT [SIGN COMPARSN PRIM/SCND]
109	$006D	191	$00BF	DUPLICATE COPY MANTISSA SIGN [LOW ORDER ROUNDING BYTE-PRM ACML]
110-111	$006E-$006F	192-193	$00C0=$00C1	POINTER TO ASCII REP OF FAC IN CONVERSION ROUTINE LO;HI
111......................................				[CASSETTE BUFFER LENGTH/TAYLOR CONSTANT POINTER]
112-117	$0070-0087	194-199	$00C2-$00D9	CHARGET RAM CODE GETS NEXT CHARACTER FROM BASIC TEXT
118	$0070-$0087	200	$00C2-$00D9	CHRGET RAM CODE TEGETS CURRENT CHARACTERS
119-120	$0070-$0087	201-202	$00C2-$00D9	POINTER TO SOURCE TEXT LO;HI
121-140	$0088-$008C	203-223		NEXT RANDOM NUMBER IN STORAGE / OR

136-140	$0088-008C			NEXT RANDOM IN STORAGE
141-143	$008D-$008F	512-514	$0200-$0202	24 HOUR CLOCK IN 1/60 SEC [CLOCK THAT INCREMENTS 60 PER SEC]
144-145	$0090-$0091	537-538	$0219-$021A	IRQ RAM VECTOR LO;HI [BREAK INTERRUPT VECTOR]
146-147	$0092-0093	539-540	$021B-$021C	BRK INSTRUCTION RAM VECTOR LO;HI [BREAK INTERRUPT VECTOR]
148	$0094-$0095	.....	.....	NMI RAM VECTOR
150	$0096	524	$020C	I/O OPERATION STATUS BYTE [STATUS (ST)]
151	$0097	515	$0203	LAST KEY INDEX [WHICH KEY DEPRSD/255=NO KEY] [MATRIX ROW COLUMN]
152	$0098	516	$0204	SHIFT FLAG (0=NO SHIFT/1=SHIFT)[SHIFT KEY 1 IF DEPRESSED]
153-154	$0099-$009A	517-518	$0205-$0206	CORRECTION FACTOR FOR CLOCK LSB; MSB [CLOCK INCREMENTS 30/SEC]
155.	$009B	521	$0209	DUPLICATE OF 59410(NEW?)BOTTOM ROW KEYS [KEYSWITCH PIA FLAGS]
157	$009D	523	$020B	FLAG <> MEANS VERIFY NOT LOAD INTO MEMORY [LOAD=0*VERIFY=1]
158	$009E	525	$020D	INDEX INTO KEYSTROKE BUFFER [# OF CHARACTERS KYSTRK BUFFER]

159	$009F	526	$020E	FLAG TO INDICATE REVERSE FIELD ON
160-166	$00A0-$00A6	.....	.....	UNUSED
167	$00A7	.....	.....	CURSOR ON FLAG
168	$00A8	549	$0225	COUNT OF JIFFIES TO BLINK CURSOR [CURSOR TIMING CNTDN][POKE=1]
169	$00A9	550	$0226	SCREEN VALUE OF CHARACTER UNDER CURSOR
170	$00AA	551	$0227	CHAR SAVED DURING BLNK/CRSR ON/OFF FLG [CURSR BLNK FLG/POKE=0]
171-173	$00AB-$00AB	.....	.....	UNUSED
174	$00AE	610	$0262	POINTER INTO LOGICAL FILE TABLE [NUMBER OF OPEN FILES]
175	$00AF	611	$0263	DEFAULT INPUT INTO DEVICE # [NORMALLY=0]
176	$00B0	612	$026A	DEFAULT OUTPUT DEVICE # [OUTPUT TO CMD DEVICE, NORMALLY=3]
177	$00B1	613	$0265	TAPE VERTICAL PARITY/COMPUTATION OF PARITY ON CASSETTE WRITE
178-185	$00B2-$00B9	.....	.....	UNUSED
180	$00B4	229-233		GENERAL PURPOSE AND ADDRESS DIRECT LO;HI(ALSO LOCATN 201-204)
181	$00B5	616	$0268	TAPE BUFFER ITEM COUNTER [POINTER IN FILE NAME TRANSFER]
184	$00B8	621	$026D	[COUNT OF REDUNDANT TAPE BLOCKS]
186	$00BA	624	$0270	SYNC ON TAPE HEADER COUNT/COUNTDOWN SYNC ON CASSETTE WRITE
187-188	$00BB-$00BC	625-626	$0271-$0272	POINTER TO ACTIVE CASSETTE/OR
188....................................				INDEX NEXT CHARACTER IN/OUT TAPE BUFFER #1; #2[187 FOR #1/188 - #2]
189	$00BD	627	$0273	COUNTDOWN SYNCHRONISATION - CASSETTE READ READ[LEAD CNTR/PASS 1/2]
190	$00BE	628	$0274	FLAG TO INDICATE BIT/BYTE TAPE ERROR[WRITE NEW BYTE]
191	$00BF	629	$0275	FLAG TO INDICATE TAPE ROUTINE READING SHORTS [WRITE START BIT]
192-193	$00C0-$00C1	630-631	$00C0-$00C1	INDEX TO ADDRESSES TO CORRECT ON TAPE READ PASS 1;PASS 2
193....................................				[192 FOR PASS 1 ERROR LOG POINTR/193 FOR PASS 2 ERR LOG PNTR]
194	$00C2	632	$0278	FLAG-CASSETTE READ-TELLS CURRENT FNCTN-CNTDWN,READ
195	$00C3	633	$0279	COUNT OF SECONDS OF SHORTS TO WRITE BEFORE DATA [CHECKSUM]

196-197	$00C4-$00C5	224-225	$00E0-$00E1	POINTER TO CURSOR POSITION [SCREEN POSITION ON LINE]
198	$00C6	226	$00E2	COLUMN POSITION OF CURSOR [POSITION OF CURSOR ON LINE][0-79]
199-200	$00C7-$00C8	227-228	$00E3-$00E4	LOAD START ADDRESS LO;HI [UTILITY POINTER]
200				[TAPE BUFFER, SCROLLING][INVERSE VIDEO CURSOR=1]
201-202	$00C9-$00CA	229-233	$00E5-$00E9	PRINT LOAD END ADDRESS LO;HI (INCLUDING LOCATION 180
202				[END OF CURRENT PROGRAM/TAPE END ADDRESS]
205	$00CD	234	$00EA	FLAG FOR QUOTE MODE ON/OFF [DIRECT/PROGRAMMED CURSOR*0=DIRECT]
206-208	$00CE-$00D0	.....	.....	UNUSED
209	$00D1	238	$00EE	CURRENT FILE NAME LENGTH [NUMBER OF CHAR-ACTERS IN FILE NAME
210	$00D2	239	$00EF	CURRENT FILE LOGICAL ADDRESS [GPIB FILE #]
211	$00D3	240	$00F0	CURRENT PRIMARY ADDRESS FILE [FILE COMMAND (FROM OPEN)][GPIB CMND]
212-213	$00D4-$00D5	241-242	$00F1-$00F2	CURRENT SECONDARY ADDRESS [212 DEVC #][213 MAX LINE LNTH 40/80]
214-215	$00D6-$00D7	243-244	$00F3-$00F4	START OF CURRENT TAPE BUFFER POINTER LO;HI
216	$00D8	245	$00F5	CURRENT SCREEN LINE # [LINE WHERE CURSOR LIVES]
217	$00D9	246	$00F6	DATA TEMPORARY FOR I/O [LAST KEY HIT(ASCII)* BUFFER CHECKSUM]
218-219	$00DA-$00DB	249-250	$00F9-$00FA	POINTER TO CURRENT FILE NAME LO;HI [FILE NAME POINTER]
220-221	$00DC-$00DD	.....	.....	UNUSED
222	$00DE	.....	.....	CASSETTE READ BLOCK COUNT
223	$00DF	.....	.....	UNUSED
224-248	$00E0-$00F8	553-577	$0229-$0241	TABLE OF LSB START ADDRESSES OF VIDEO DISPLAY LINES (25)
225				[SCREEN LINE WRAP TABLE]
249-250	$00F9-$00FA	519-520	$0207-$0208	INTERRUPT DRIVER FLAG FOR CASSETTE # 1 SWITCHES; # 2 SWITCHES

250				[249 FOR CASSETTE #1 ON][250 FOR CASSETTE #2 ON]
251-252	$00FB-$00FC	247-248	$00F7-$00F8	POINTER TO START LOC FOR O.S. LO;HI [TAPE START ADDRESS*TAPE PNT]
252				[POINTER TO PROGRAM DURING VERIFY, LOAD]
512-591	$0020-$0250	10-89	$00A0-$0059	BASIC INPUT BUFFER (80 BYTES)[NUMBER OF ARRAY SUBSCRIPTS]
513				512-513 IS THE PROGRAM COUNTER
514				514 IS PROCESSOR STATUS
515				515 IS ACCUMULATOR
516				516 IS X INDEX
517				517 IS Y INDEX
518				518 IS STACK POINTER
519-520				519-520 IS USER MODIFIABLE IRQ
593-602	$0251-$025A	578-587	$0242-$024B	LOGICAL FILE NUMBERS [LOGICAL NUMBERS OF OPEN FILES]
603-612	$025B-$0264	588-597	$024C-$0255	PRIMARY DEVICE NUMBERS [DEVICE #S OF OPEN FILES]
613-622	$0265-$026E	598-609	$0256-$0261	TABLE OF SECONDARY ADDRESSES [COMMAND/ SECONDARY ADRS OPEN FIL]
623-632	$026F-$0278	527-536	$020F-$0218	INTERRUPT DRIVEN KEYSTROKE BUFFER
634-825	$027A-$0339			BUFFER FOR CASSETTE #1 (192 BYTES)
826-1017	$033A-$03F9	826-1017	$033A-$03F9	BUFFER FOR CASSETTE #2 (192 BYTES)

The Independent PET Users' Group (South) has been re-named, and is now known as SUPA (Southern Users of PETs Association). Their former position as a regional Group for IPUG is now terminated.

Subscription rate for 12 months is now 5.00. Applicants should contact the Membership Secretary:

Howard Pilgrim
42 Compton Road
Brighton
Sussex
BN1 5AN

* * * * * * * * * *

PET SPEAKS

Petsoft have released a 10 'Talking Claculator' program which generates synthetic speech via an inexpensive soundbox connected to the user port of the Commodore PET microcomputer.

Once loaded, the program causes the name of each numeric or function key to be generated as synthetic speeck, followed by the result of the calculation. The representation of a calculator on screen allows the calculation to be tracked visually as well.

As each key is depressed, the computer responds with the words: "Six...divided...by...four...equals...one ...point...five".

The 'Talking Calculator' is available, price 10 + VAT from PET dealers, or by mail order direct from ACT Petsoft, 66-68 Hagley Road, Edgbaston, Birmingham, B16 8PF. A User Port Speech and Music Generator with volume control and connectors is also on sale priced at 27 + VAT

SPACE INVADERS ZAP PET

I, for one, have worn the 'A' key out on my PET. Here is an ideal solution by G. Luxford .....

"I have found the SPACE INVADER program to be a knock-out, but the key for beam firing, "A" is coming in for severe pounding, particularly when other over enthusiastic hands get onto the keyboard.

With the new ROM version a temporary solution is to shift the commands onto different keys. This can be achieved by POKE1409,6 after loading but prior to running the program, to shift commands to Z, 1 and 3. This location can alternatively be POKEd with any value from 0 to 9, except 4 (present values) to select other options for the keyboard. Another option is to POKE1938 with 2, 4, 8 or 16 to shift the beam fixing key along the key row.

A better solution is to shift the control to the user port input. This can be achieved by POKE1414,79 and POKE1417,79. Control is then by switch closers to the ground line, pin N of PA0 on pin C to fire beam, PA7 on pin L to move the laser base right and PA6 on pin K to move the laser base left. You can now use large hefty buttons or a joy stick control with complete immunity to keyboard damage by over-excited saucer shooters.

You may SAVE the program either by BASIC without a name or by TIM monitor, from $0400 to $2000.

The program is also being modified to give the option of any alpha key to fire the beam, any numeric key to shift the laser base and the option of control by the user port. This requires appreciably more patching and is not yet fully debugged.

When completed I will send details with a revised program to CPUCN.

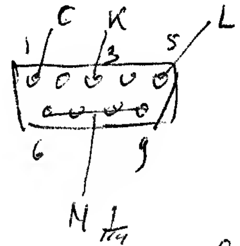
I regret not being of direct help to old ROM users, having recently become a new ROMer, but can only suggest they try the recommended program mods and see what happens. There seems a good chance this will work."

The result of the PET Show Prize Draw.

The draw was made by Kit Spencer, in the presence of students on a BASIC for Beginners course at the Skyway Hotel on 17 June.

The winner of the 95 software prize is:-

E Ramsden
The Country House
123 Greenland Road
Hemel Hempstead
HK1 1RT



1 = green = C
3 = blue = K
5 = [illegible] = L
Sound = wire = N